SPIRE User's Manual

version 0.4.dev, Document Number: Not Yet Assigned
12 June 2009

SPIRE User's Manual

Table of Contents

Preface xii
1. Versioning xii
1.1. Changelog xii
1. Introduction 1
1.1. Scope of this User's Manual 1
1.2. The SPIRE Pipeline 1
1.3. Installation and Startup of HIPE 2
2. Accessing the Raw Data 3
2.1. Accessing the Data 3
3. Processing the Raw Data: The Engineering Conversion 4
3.1. Introduction 4
3.2. Running the *Engineering Conversion* Pipeline 5
3.2.1. Module Description 5
3.2.2. Input Data Products 5
3.2.3. Output Data Products 5
3.2.4. Input Calibration Products 5
3.2.5. How to use the *Engineering Conversion* Pipeline 5
3.2.6. Parameter Options 5
3.2.7. Examples 6
3.2.8. Error messages 7
3.3. The *Engineering Conversion* Pipeline Step-by-Step: Converting Level 0 Products to Level 0.5 format 7
3.3.1. Module Description 7
3.3.2. Input Data Products 7
3.3.3. Output Data Products 7
3.3.4. Input Calibration Products 7
3.3.5. How to use the *Format Conversion* Module 7
3.3.6. Parameter Options 7
3.3.7. Examples 7
3.3.8. Error messages 8
3.4. The *Engineering Conversion* Pipeline Step-by-Step: Flagging out of range detectors 8
3.4.1. Module Description 8
3.4.2. Input Data Products 8
3.4.3. Output Data Products 8
3.4.4. Input Calibration Products 8
3.4.5. How to use the *Flag Detectors* Module 8
3.4.6. Parameter Options 8
3.4.7. Examples 8
3.4.8. Error messages 9
3.5. The *Engineering Conversion* Pipeline Step-by-Step: Masking Bad Detector Channels 9
3.5.1. Module Description 9
3.5.2. Input Data Products 9
3.5.3. Output Data Products 9
3.5.4. Input Calibration Products 9
3.5.5. How to use the *Mask Bad Channels* Module 9
3.5.6. Parameter Options 9
3.5.7. Examples 9
3.5.8. Error messages 10
3.6. The *Engineering Conversion* Pipeline Step-by-Step: Time Conversion 10
3.6.1. Module Description 10
3.6.2. Input Data Products 10
3.6.3. Output Data Products 10
3.6.4. Input Calibration Products 10

3.6.5. How to use the `TimeConvReordTask` 10
3.6.6. Parameter Options 10
3.6.7. Examples 10
3.6.8. Error messages 11
3.7. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the JFET Voltages 11
3.7.1. Module Description 11
3.7.2. Input Data Products 11
3.7.3. Output Data Products 11
3.7.4. Input Calibration Products 11
3.7.5. How to use the *Calculate JFET Voltages* Module 12
3.7.6. Parameter Options 12
3.7.7. Examples 12
3.7.8. Error messages 12
3.8. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the bolometer RMS voltage and resistance 13
3.8.1. Module Description 13
3.8.2. Input Data Products 13
3.8.3. Output Data Products 13
3.8.4. Input Calibration Products 13
3.8.5. How to use the *Calculate JFET Voltages* Module 13
3.8.6. Parameter Options 13
3.8.7. Examples 13
3.8.8. Error messages 14
3.9. The *Engineering Conversion* Pipeline Step-by-Step: Adding Pointing Meta-Data Information to the Data 14
3.9.1. Module Description 14
3.9.2. Input Data Products 14
3.9.3. Output Data Products 15
3.9.4. Input Calibration Products 15
3.9.5. How to use the *Add Pointing Meta-Data* Module 15
3.9.6. Input Control Parameters 15
3.9.7. Examples 15
3.9.8. Error messages 15
4. Processing Data with the Photometer Large Scan Map Pipeline 16
4.1. Introduction 16
4.2. Conversion of the BSM telemetry into Angles on the Sky 17
4.2.1. Module Description 17
4.2.2. Input Data Products 17
4.2.3. Output Data Products 17
4.2.4. Input Calibration Products 17
4.2.5. How to use the *Compute BSM Angles* Module 17
4.2.6. Parameter Options 17
4.2.7. Examples 18
4.2.8. Error messages 18
4.3. Deglitching the Timeline Data 19
4.3.1. Module Description 19
4.3.2. Input Data Products 19
4.3.3. Output Data Products 19
4.3.4. Input Calibration Products 19
4.3.5. How to use the *First Level Deglitching* Module 19
4.3.6. Parameter Options 19
4.3.7. Examples 20
4.3.8. Error messages 20
4.4. Removing Electrical Crosstalk from the Timeline Data 20
4.4.1. Module Description 20
4.4.2. Input Data Products 21
4.4.3. Output Data Products 21

4.4.4. Input Calibration Products 21
4.4.5. How to use the *Remove Electrical Crosstalk* Module 21
4.4.6. Parameter Options 21
4.4.7. Examples 21
4.4.8. Error messages 21
4.5. Correcting for Electrical Filter Response 21
4.5.1. Module Description 21
4.5.2. Input Data Products 22
4.5.3. Output Data Products 22
4.5.4. Input Calibration Products 22
4.5.5. How to use the *Correction for Electrical Filter Response* Module 22
4.5.6. Parameter Options 22
4.5.7. Examples 22
4.5.8. Error messages 22
4.6. Converting the Detector Timelines into Flux Units 23
4.6.1. Module Description 23
4.6.2. Input Data Products 23
4.6.3. Output Data Products 23
4.6.4. Input Calibration Products 23
4.6.5. How to use the *Flux Conversion* Module 23
4.6.6. Parameter Options 23
4.6.7. Examples 23
4.6.8. Error messages 23
4.7. Removing Correlated Noise from the Detector Timelines 24
4.7.1. Module Description 24
4.7.2. Input Data Products 24
4.7.3. Output Data Products 24
4.7.4. Input Calibration Products 24
4.7.5. How to use the *Remove Correlated Noise* Module 24
4.7.6. Parameter Options 24
4.7.7. Examples 24
4.7.8. Error messages 25
4.8. Correcting for the Bolometer Time Response 25
4.8.1. Module Description 25
4.8.2. Input Data Products 25
4.8.3. Output Data Products 25
4.8.4. Input Calibration Products 25
4.8.5. How to use the *Bolometer Time Response Correction* Module 25
4.8.6. Parameter Options 25
4.8.7. Examples 26
4.8.8. Error messages 26
4.9. Removing the Effects of Optical Crosstalk from the Data 26
4.9.1. Module Description 26
4.9.2. Input Data Products 26
4.9.3. Output Data Products 26
4.9.4. Input Calibration Products 26
4.9.5. How to use the *Optical Crosstalk Removal* Module 26
4.9.6. Parameter Options 26
4.9.7. Examples 27
4.9.8. Error messages 27
4.10. Adding Positional Information (WCS) to the Data 27
4.10.1. Module Description 27
4.10.2. Input Data Products 27
4.10.3. Output Data Products 27
4.10.4. Input Calibration Products 27
4.10.5. How to use the *Associate Sky Position* Module 27
4.10.6. Parameter Options 27
4.10.7. Examples 27

4.10.8. Error messages 27
4.11. Converting the On-Board Time to International Time Standards 28
4.11.1. Module Description 28
4.11.2. Input Data Products 28
4.11.3. Output Data Products 28
4.11.4. Input Calibration Products 28
4.11.5. How to use the *Time Correction* Module 28
4.11.6. Parameter Options 28
4.11.7. Examples 28
4.11.8. Error messages 28
4.12. Creating Image Scan-Maps 28
4.12.1. Module Description 28
4.12.2. Input Observational Data Products 29
4.12.3. Output Data Products 29
4.12.4. Input Calibration Products Required 29
4.12.5. How to use the *Map Making* Module 29
4.12.6. Parameter options 29
4.12.7. Examples 30
4.12.8. Error messages 30
4.13. Point Source Extraction 30
4.13.1. Module Description 30
4.13.2. Input Observational Data Products 30
4.13.3. Output Data Products 30
4.13.4. Input Calibration Products Required 30
4.13.5. How to use the *Point Source Extraction* Module 31
4.13.6. Parameter options 31
4.13.7. Examples 31
4.13.8. Error messages 31
5. Processing Data with the Photometer Jiggle Map Pipeline 32
5.1. Introduction 32
5.2. Conversion of the BSM telemetry into Angles on the Sky 34
5.2.1. Module Description 34
5.2.2. Input Data Products 34
5.2.3. Output Data Products 34
5.2.4. Input Calibration Products 34
5.2.5. How to use the *Compute BSM Angles* Module 34
5.2.6. Parameter Options 34
5.2.7. Examples 34
5.2.8. Error messages 35
5.3. Extracting the Chop and Jiggle Positions from the BSM Timeline 35
5.3.1. Module Description 35
5.3.2. Input Data Products 36
5.3.3. Output Data Products 36
5.3.4. Input Calibration Products 36
5.3.5. How to use the *Extract Chop and Jiggle Positions* Module 37
5.3.6. Parameter Options 37
5.3.7. Examples 37
5.3.8. Error messages 37
5.4. Deglitching the Timeline Data 37
5.4.1. Module Description 37
5.4.2. Input Data Products 38
5.4.3. Output Data Products 38
5.4.4. Input Calibration Products 38
5.4.5. How to use the *First Level Deglitching* Module 38
5.4.6. Parameter Options 38
5.4.7. Examples 39
5.4.8. Error messages 39
5.5. Removing Electrical Crosstalk from the Timeline Data 39

5.5.1. Module Description 39
5.5.2. Input Data Products 39
5.5.3. Output Data Products 39
5.5.4. Input Calibration Products 39
5.5.5. How to use the *Remove Electrical Crosstalk* Module 40
5.5.6. Parameter Options 40
5.5.7. Examples 40
5.5.8. Error messages 40
5.6. Converting the Detector Timelines into Flux Units 40
5.6.1. Module Description 40
5.6.2. Input Data Products 40
5.6.3. Output Data Products 40
5.6.4. Input Calibration Products 41
5.6.5. How to use the *Flux Conversion* Module 41
5.6.6. Parameter Options 41
5.6.7. Examples 41
5.6.8. Error messages 41
5.7. De-Modulating the Timeline Data 41
5.7.1. Module Description 41
5.7.2. Input Observational Data Products 41
5.7.3. Output Data Products 42
5.7.4. Input Calibration Products Required 42
5.7.5. How to use the *Demodulation* Module 42
5.7.6. Parameter options 42
5.7.7. Examples 42
5.7.8. Error messages 42
5.8. Averaging the Data of All Jiggle Positions (and Second Level Deglitching) 42
5.8.1. Module Description 42
5.8.2. Input Data Products 43
5.8.3. Output Data Products 43
5.8.4. Input Calibration Products 43
5.8.5. How to use the *Second Level Deglitching and Averaging* Module 43
5.8.6. Parameter Options 43
5.8.7. Examples 43
5.8.8. Error messages 43
5.9. De-Nodding Observations 43
5.9.1. Module Description 43
5.9.2. Input Observational Data Products 43
5.9.3. Output Data Products 43
5.9.4. Input Calibration Products Required 44
5.9.5. How to use the *Denodding* Module 44
5.9.6. Parameter options 44
5.9.7. Examples 44
5.9.8. Error messages 44
5.10. Averaging the Data Taken from Different Nod Cycles 44
5.10.1. Module Description 44
5.10.2. Input Data Products 45
5.10.3. Output Data Products 45
5.10.4. Input Calibration Products 45
5.10.5. How to use the *Average Nod Cycles* Module 45
5.10.6. Parameter Options 45
5.10.7. Examples 45
5.10.8. Error messages 45
5.11. Removing the Effects of Optical Crosstalk from the Data 45
5.11.1. Module Description 45
5.11.2. Input Data Products 45
5.11.3. Output Data Products 45
5.11.4. Input Calibration Products 45

5.11.5. How to use the *Optical Crosstalk Removal* Module 46
5.11.6. Parameter Options 46
5.11.7. Examples 46
5.11.8. Error messages 46
5.12. Converting the On-Board Time to International Time Standards 46
5.12.1. Module Description 46
5.12.2. Input Data Products 46
5.12.3. Output Data Products 46
5.12.4. Input Calibration Products 46
5.12.5. How to use the *Time Correction* Module 46
5.12.6. Parameter Options 46
5.12.7. Examples 46
5.12.8. Error messages 47
5.13. Creating Image Maps from the Jiggle Observations 47
5.13.1. Module Description 47
5.13.2. Input Observational Data Products 47
5.13.3. Output Data Products 47
5.13.4. Input Calibration Products Required 47
5.13.5. How to use the *Map Making* Module 47
5.13.6. Parameter options 48
5.13.7. Examples 48
5.13.8. Error messages 48
5.14. Calculating the Flux and Position of a Source 48
5.14.1. Module Description 48
5.14.2. Input Data Products 48
5.14.3. Output Data Products 48
5.14.4. Input Calibration Products Required 49
5.14.5. How to use the *Point Source Flux Density and Position* Module 49
5.14.6. Parameter options 49
5.14.7. Examples 49
5.14.8. Error messages 49
6. Processing Data with the Spectrometer Pipeline 50
6.1. Introduction 50
6.2. Conversion of the BSM telemetry into Angles on the Sky 52
6.2.1. Module Description 52
6.2.2. Input Data Products 52
6.2.3. Output Data Products 52
6.2.4. Input Calibration Products 52
6.2.5. How to use the *Compute BSM Angles* Module 52
6.2.6. Parameter Options 52
6.2.7. Examples 53
6.2.8. Error messages 53
6.3. Deglitching the Detector Timelines 54
6.3.1. Module Description 54
6.3.2. Input Data Products 54
6.3.3. Output Data Products 54
6.3.4. Input Calibration Data Products 54
6.3.5. How to use the *First Level Deglitching* Module 54
6.3.6. Parameter Options 54
6.3.7. Examples 55
6.3.8. Error messages 56
6.4. Correcting the Detector Timelines for Electrical Crosstalk 56
6.4.1. Module Description 56
6.4.2. Input Data Products 57
6.4.3. Output Data Products 57
6.4.4. Input Calibration Products 57
6.4.5. How to use the *Electrical Cross Talk Correction* Module 57
6.4.6. Parameter Options 57

6.4.7. Examples 57
6.4.8. Error messages 57
6.5. Correcting for Clipped Data 58
6.5.1. Module Description 58
6.5.2. Input Data Products 58
6.5.3. Output Data Products 58
6.5.4. Input Calibration Products 58
6.5.5. How to use the *Clipping Correction* Module 58
6.5.6. Parameter Options 58
6.5.7. Examples 58
6.5.8. Error messages 59
6.6. Correcting for Time Domain Phase Shifts in the Data 59
6.6.1. Module Description 59
6.6.2. Input Observational Data Products 59
6.6.3. Output Data Products 59
6.6.4. Input Calibration Data Products 60
6.6.5. How to use the *Time Domain Phase Correction* Module 60
6.6.6. Control parameters 60
6.6.7. Examples 60
6.6.8. Error messages 62
6.7. Applying the Non-Linearity Correction to the Data 62
6.7.1. Module Description 62
6.7.2. Input Data Products 63
6.7.3. Output Data Products 63
6.7.4. Input Calibration Products 63
6.7.5. How to use the *Non-Linearity Correction* Module 63
6.7.6. Parameter Options 63
6.7.7. Examples 63
6.7.8. Error messages 63
6.8. Removing Correlated Noise due to bath temperature fluctuations from the Data 64
6.8.1. Module Description 64
6.8.2. Input Data Products 64
6.8.3. Output Data Products 64
6.8.4. Input Calibration Products 64
6.8.5. How to use the *Bath Temperature Fluctuation Correction* Module 64
6.8.6. Parameter Options 64
6.8.7. Examples 64
6.8.8. Error messages 65
6.9. Creating the Interferograms from the Timeline Data 66
6.9.1. Module Description 66
6.9.2. Input Data Products 66
6.9.3. Output Data Products 66
6.9.4. Input Calibration Products 67
6.9.5. How to use the *Interferogram Creation* Module 67
6.9.6. Control parameters 67
6.9.7. Examples 67
6.9.8. Error messages 68
6.10. Removing the Telescope and Calibration Source Background from the Data 69
6.10.1. Module Description 69
6.10.2. Input Data Products 69
6.10.3. Output Data Products 69
6.10.4. Input Calibration Products 70
6.10.5. How to use the *SCAL and Telescope Correction* Module 70
6.10.6. Parameter Options 70
6.10.7. Examples 70
6.10.8. Error messages 70
6.11. Removing the Baseline from the interferograms 70
6.11.1. Module Description 70

6.11.2. Input Data Products 71
6.11.3. Output Data Products 71
6.11.4. Calibration Data Products 71
6.11.5. How to use the *Baseline Correction* Module 71
6.11.6. Parameter Options 71
6.11.7. Examples 72
6.11.8. Error messages 72
6.12. Removing Glitches from the Interferograms 73
6.12.1. Module Description 73
6.12.2. Input Data Products 73
6.12.3. Output Data Products 73
6.12.4. Input Calibration Products 73
6.12.5. How to use the *Second-Level Deglitching* Module 73
6.12.6. Parameter Options 73
6.12.7. Examples 74
6.12.8. Error messages 75
6.13. Phase Correction 75
6.13.1. Module Description 75
6.13.2. Input Data Products 76
6.13.3. Output Data Products 76
6.13.4. Input Calibration Products 76
6.13.5. How to use the *Phase Correction* Module 76
6.13.6. Control parameters 76
6.13.7. Examples 77
6.13.8. Error messages 78
6.14. Applying an Apodization Function to an Interferogram 79
6.14.1. Module Description 79
6.14.2. Input Data Products 79
6.14.3. Output Data Products 79
6.14.4. Input Calibration Data Products 79
6.14.5. How to use the *Apodization* Module 79
6.14.6. Parameter Options 79
6.14.7. Examples 80
6.14.8. Error messages 82
6.15. Producing Spectra from the Interferograms (Fourier Transform) 83
6.15.1. Module Description 83
6.15.2. Input Data Products 83
6.15.3. Output Data Products 83
6.15.4. Input Calibration Products 83
6.15.5. How to use the *Fourier Transform* Module 83
6.15.6. Parameter Options 83
6.15.7. Examples 84
6.15.8. Error messages 84
6.16. Spectrum Flux Calibration 84
6.16.1. Module Description 84
6.16.2. Input Data Products 84
6.16.3. Output Data Products 84
6.16.4. Input Calibration Products 84
6.16.5. How to use the *Flux Conversion* Module 85
6.16.6. Parameter Options 85
6.16.7. Examples 85
6.16.8. Error messages 85
6.17. Removing any Optical Crosstalk from the Spectra 85
6.17.1. Module Description 85
6.17.2. Input Data Products 85
6.17.3. Output Data Products 85
6.17.4. Input Calibration Products 86
6.17.5. How to use the *Remove Optical Crosstalk* Module 86

6.17.6. Parameter Options 86
6.17.7. Examples 86
6.17.8. Error messages 86
6.18. Averaging Spectra to Produce One Final Spectral Product 86
6.18.1. Module Description 86
6.18.2. Input Data Products 86
6.18.3. Output Data Products 87
6.18.4. Input Calibration Data Products 87
6.18.5. How to use the *Spectral Averaging* Module 87
6.18.6. Parameter Options 87
6.18.7. Examples 87
6.18.8. Error messages 88

Preface

1. Versioning

On the front page of this manual is a version number made of three digits. The first two digits follow a traditional versioning system (0.1, 0.2, ...), and the changes introduced with each version are detailed below. The third digit is the SPIRE build number to which each edition of the manual is associated. Also shown on the front page is the date of publication of the manual.

1.1. Changelog

The following was changed for v0.4

- SPIRE pipline section updated
- Flowcharts updated
- Photometer Pipeline Section Updated following Science Validation Review
- Spectrometer Pipeline Section Updated following Science Validation Review

The following was changed for v0.3

- Original UM was replaced with sections from the HOWTOs, plus the SPIRE Pipeline User's Guide, which will form the basis of the UM.

The following was changed for v0.2

- Added this preface.
- *Chapter 1*: Added workaround to reintroduce SPIRE specific help into QLA, including context-sensitive help.
- *Chapter 2*: Added the preexisting QLA manual as a new chapter.

The following was changed for v0.1

- First version of the manual, incudes the SPIRE DP Quick Setup guide as its only chapter.

Chapter 1. Introduction

1.1. Scope of this User's Manual

This document explains the SPIRE pipeline and its use for reduction of photometer and spectrometer data using the standard Observatory Functions (POF, SOF Astronomical Observing Templates, AOTs). This document is aimed at the astronomical user and calibration scientists who may wish to use the pipeline from a scientific viewpoint. This document is not meant for developers and those interested in the details of the pipeline module algorithms and processes.

This document provides the procedures to install and set up the data reduction environment (Herschel Common Science System, HCSS). A broad overview of the pipeline is given followed by detailed information of the individual pipeline steps including parameters and settings. The pipeline corresponding to each AOT is discussed. Worked examples are also provided.

1.2. The SPIRE Pipeline

The SPIRE pipeline is a part of the general Development Pipeline of the Herschel Common Science System (HCSS) being developed by the Herschel Science Center (HSC) and Herschel Instrument Control Centres (ICCs) to provide the complete software system for the Herschel Observatory mission. The entire Development Pipeline is written in Java and scripted in Jython and is a fully supported stand alone system. The pipeline can be run via scripts either end-to-end or stepwise. In addition to the standard product pipeline processing the Herschel Data Processing system will also provide the tools to interactively reduce the data through Graphical User Interfaces provided by the Herschel Interactive Processing Environment (HIPE) The pipeline supports processing for 6 different AOTs. In addition each AOT may have independent parameters to further refine the type of observation to be carried out.

An overview of the SPIRE pipline is shown in the figure below. The raw telemetry data has already been processed into the basic Level 0 Raw data format by a pre-processing step during the Operational Day Processing. The SPIRE pipeline as delivered to a user takes these Level 0 products as input. The Level 0 to Level 0.5 processing is referred to as the engineering conversion or "common pipeline" and takes the raw ADU in the timeline data and converts them to meaningful units (e.g. Volts, etc). The next step is the AOT specific processing (large map, small map, point source or spectrometer pipelines) which will produce calibrated timelines as Level 1 products. The advanced processing steps will take the Level 1 products and produce maps, spectral cubes and point source flux estimations. Quality control will be handelled by ESA rather than the ICC.

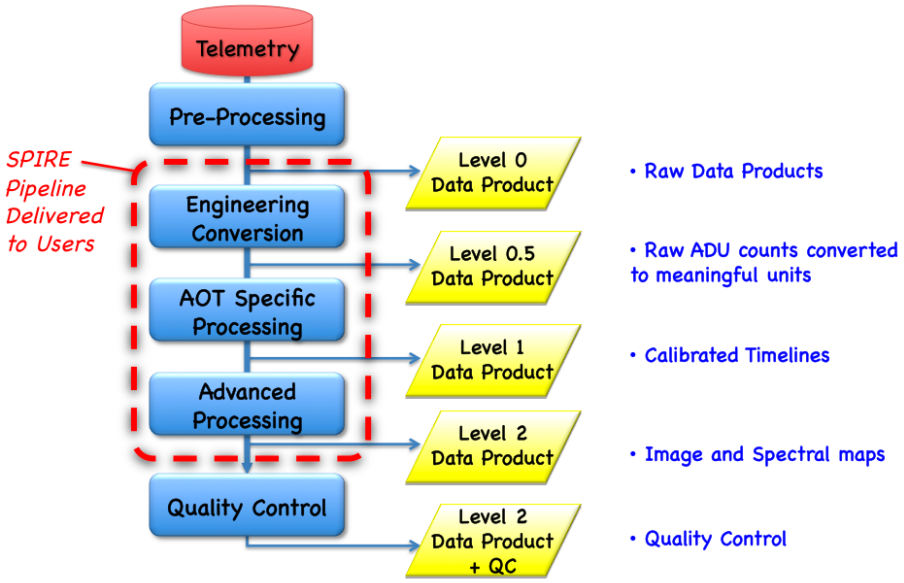

Figure 1.1. An overview of the SPIRE pipeline from the raw telemetry to the fully processed products.

1.3. Installation and Startup of HIPE

Data reduction for SPIRE datasets is performed using the Herschel Interactive Processing Environment (HIPE). HIPE is part of the Herschel Data Processing system and can be installed with the software installer (see Herschel Science Centre website). Software can be run on a server or individual workstation running Windows XP, Linux or Solaris. The minimum recommended system is Windows/Linux 32-bit w/1GB RAM or 64-bit W/Lin/Mac w/1GB RAM; Browsers for use with the systemm (including download) IE 6+ , Netscape 7+, Mozilla (Firefox) 1.5+, Safari (Mac). The system is Java based and requires Java 1.6. General Java scripts can be run on the system. Installation instructions are provided at the bottom of the FTP page.

Once the software is installed, HIPE can be started by several means. Using Windows, Herschel software can be started under the "Start" menu after a standard installation. Alternatively, HIPE can be started from a command line.

```
hipe
```

For more details on HIPE, please review the HIPE Introduction section in the Herschel DP HowTos docuument.

Chapter 2. Accessing the Raw Data

2.1. Accessing the Data

The Raw Data Extraction step is the first step in the SPIRE data processing pipeline. The purpose of this step is to extract the telemetry data (specified by a user selected range) from a database and to compile them into a set of Level-0 SPIRE data products for the use of further data processing steps.

In addition to producing the Level 0 products. It also produces the DPU reset history calibration product.

The Herschel Science Archive (HSA) is the main repository for the observational data products from Herschel. It is available via a web interface to the Herschel Science Centre. But it is also available directly from within a DP session using HIPE. For more information on acessing data sets via the HSA or through HIPE, please review the Herschel DP HowTo on accessing and retrieving data. This document is accessible through the Documentation button in HIPE.

Chapter 3. Processing the Raw Data: The Engineering Conversion

3.1. Introduction

This part of the SPIRE processing pipeline is common to both the photometer and spectrometer and converts the Level 0 raw data products into the processed Level 0.5 data products. The Pipeline to convert the raw data, i.e. the Level 0 Product, into the processed Level 0.5 Product is shown in Figure

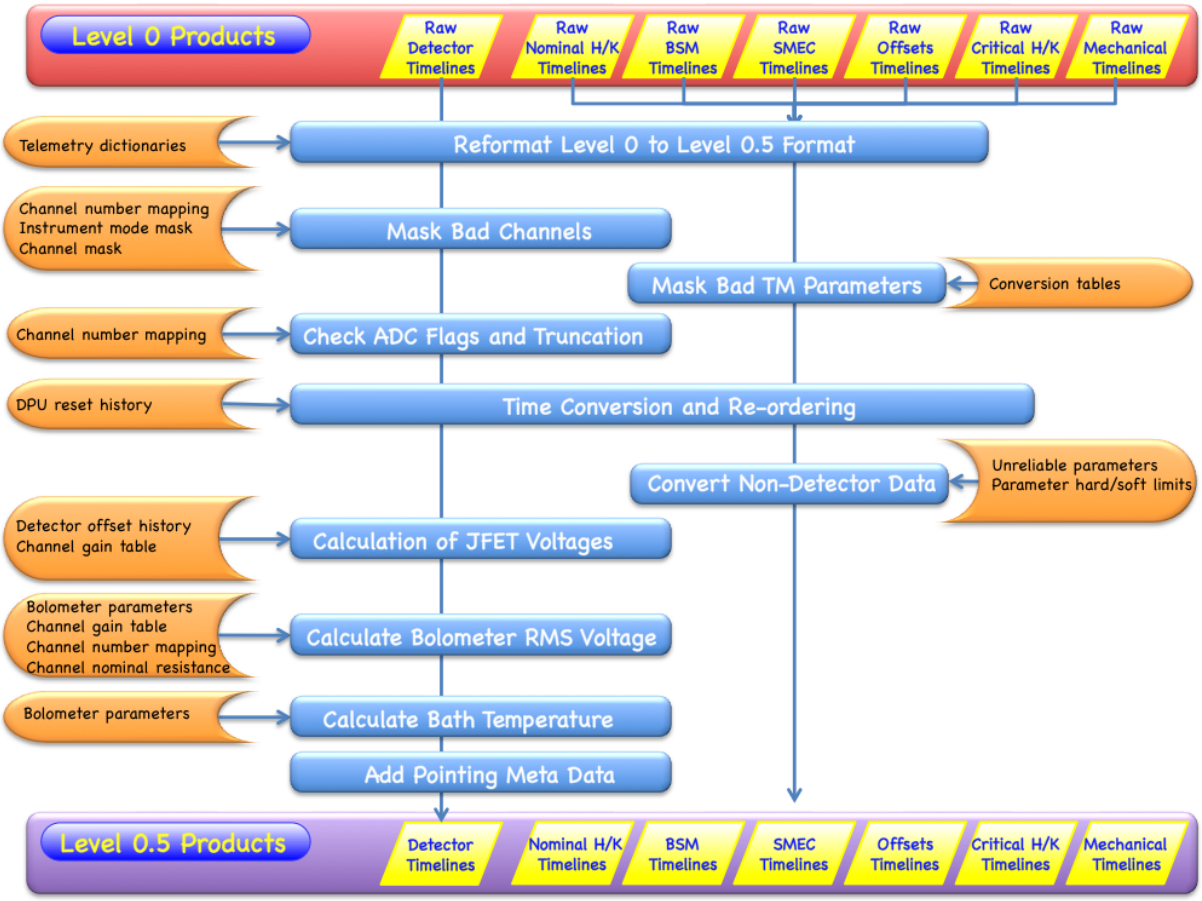

Figure 3.1. The Engineering Conversion Pipeline takes the raw Level 0 Products and converts the ADU counts into meaningful units (volts, etc).

This module is composed by several tasks. Each task can be executed as an independent step. The module provides also a main task, named `EngConversionTask`, that calls all the tasks of this module in the correct sequence.

The usage of the `EngConversionTask` instead of single internal tasks is recomanded because the processing flowchart of this step is quite complex. Moreover, the internal tasks work only on single products while the `EngConversionTask` can process a complete set of products in one single execution.

Note that an instance of all the tasks of this module is created at the start-up of HIPE, so you can use these tasks as fuctions, whitout the need of importing and initializing them.

3.2. Running the *Engineering Conversion* Pipeline

3.2.1. Module Description

The *Engineering Conversion* Pipeline is responsible for the processing of Level 0 products to Level 0.5 products.

3.2.2. Input Data Products

level0	Input Level-0 SPIRE Data Products wrapped in a Level0 Context. If you want convert in one single step all the level 0 products contained in an ObservationContex, use this parameter to give as input the Level0Context contained in the ObservationContext. This is the primary input of the task.
level0block	Input Level-0 SPIRE Data Products wrapped in a Level0BlockContext. Use this parameter if you want to give as input level 0 products belonging to a single Building Block.
rawData	Input Level-0 SPIRE Data Products as a map. Use this parameter if you want to give as input level 0 products which are not ordered in building blocks.

3.2.3. Output Data Products

Level 0.5 products	**Level 0.5 products** The output products of this module are the Level 0.5 products.

3.2.4. Input Calibration Products

cal	Spire Calibration Context. You can use this parameter to give to the task all calibration products warped in a calibration context. If a calibration product is provided using the specific parameter (see below), the corresponding product contained in the calibration context is ignored.
resetHist	DPU reset history calibration product.
offsetHist	Detector offset history calibration product.
chanGain	Channel Gain calibration product.
bolPar	Bolometer parameters calibration product.
chanMask	Channel Mask calibration product.
chanNum	Channel Number Mapping calibration product.
chanNomRes	Channel Nominal Resistances calibration product.

3.2.5. How to use the *Engineering Conversion* Pipeline

3.2.6. Parameter Options

Users can set the following parameters of the module:

useSink	Flag to use or not ProductSink. If this flag is set to true, the task will save the processed products into the ProductSink. This allows to process big observations without the need of a large memory. Note that the ProductSink shall be initialized before executing the task, if this flag is set to true.

3.2.7. Examples

In the following example, the `EngConversionTask` is used to convert all level 0 products of an ObservationContext.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>>
 IA>> level0_5=engConversion(level0=obs.level["level0"], cal=obs.calibration)
 IA>>
 IA>> obs.level["level0_5"]=level0_5  #here we put the result into the
ObservationContext
```

Since `level0` is the first parameter of the `EngConversionTask`, we can simplify the sintax in:

```
 IA>> level0_5=engConversion(obs.level["level0"], cal=obs.calibration)
```

The following example is similar to the previous but here we use a ProductSink.

```
 IA>> # import needed classes
 IA>> from herschel.ia.pal import *
 IA>> from herschel.ia.pg import ProductSink
 IA>>
 IA>> # create a ProductStorage to be used by the ProductSink
 IA>> sinkStorage=ProductStorage()
 IA>> # register a pool called "sink" where product will be stored
 IA>> sinkStorage.register(PoolManager.getPool("sink"))
 IA>> # attach the storage to the ProductSink
 IA>> ProductSink.getInstance().productStorage=sinkStorage
 IA>>
 IA>> obs=.... #obs is the ObservationContext
 IA>>
 IA>> level0_5=engConversion(obs.level["level0"], cal=obs.calibration,
useSink=True)
 IA>>
 IA>> obs.level["level0_5"]=level0_5  #here we put the result into the
ObservationContext
```

In the following example, we convert the level 0 products of a single building block and we request the task to use a specific Channel Mask calibration product.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> block=obs.level["level0"].get(obsid,bbid)
 IA>> chanMask=.....
 IA>> level0_5=engConversion(level0block=block, cal=obs.calibration,
chanMask=chanMask)
 IA>>
 IA>> obs.level["level0_5"]=level0_5  #here we put the result into the
ObservationContext
```

3.2.8. Error messages

TBW. TBW.

TBW. TBW.

3.3. The *Engineering Conversion* Pipeline Step-by-Step: Converting Level 0 Products to Level 0.5 format

3.3.1. Module Description

The *Format Conversion* Module implements the reformat of Level 0 products into Level 0.5 products format.

3.3.2. Input Data Products

rawData	Input Level-0 SPIRE Data Product to be reformatted.

3.3.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** The data is reformatted into the style of Level 0.5 products.

3.3.4. Input Calibration Products

3.3.5. How to use the *Format Conversion* Module

The `FormatConversionTask`

3.3.6. Parameter Options

Users can set the following parameters of the module:

3.3.7. Examples

In the following example, we reformat a level 0 product into level 0.5 format.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of building
block 0xA0300001
 IA>>
 IA>> pdt=formatConversion(rawData=rpdt) # or you can do also
pdt=formatConversion(rpdt)
```

3.3.8. Error messages

TBW. TBW.

TBW. TBW.

3.4. The *Engineering Conversion* Pipeline Step-by-Step: Flagging out of range detectors

3.4.1. Module Description

The *Flag Detectors* Module iimplements the checking of ADC flags and truncation..

3.4.2. Input Data Products

rawData	Input detector timeline to be processed.

3.4.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.4.4. Input Calibration Products

3.4.5. How to use the *Flag Detectors* Module

The `DetFlaggerTask` task

3.4.6. Parameter Options

Users can set the following parameters of the module:

3.4.7. Examples

In the following example, we process a PDT.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of building
block 0xA0300001
 IA>>
 IA>> pdt=formatConversion(rpdt) # reformatting
 IA>> pdt=detFlagger(pdt)
```

3.4.8. Error messages

TBW. TBW.

TBW. TBW.

3.5. The *Engineering Conversion* Pipeline Step-by-Step: Masking Bad Detector Channels

3.5.1. Module Description

The *Mask Bad Channels* Module implements the masking of bad channels.

3.5.2. Input Data Products

data	Input detector timeline to be processed.

3.5.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.5.4. Input Calibration Products

chanMask	Channel Mask calibration product.
chanNum	Channel Number Mapping calibration product.

3.5.5. How to use the *Mask Bad Channels* Module

The `MaskBadChanTask` implements the masking of bad channels.

3.5.6. Parameter Options

Users can set the following parameters of the module:

3.5.7. Examples

In the following example, we process a PDT using only the channel mask. The PDT doesn't contain disconnected channels.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of building
block 0xA0300001
 IA>>
 IA>> pdt=formatConversion(rpdt) # reformatting
 IA>> pdt=detFlagger(pdt)
 IA>> pdt=maskBadChan(pdt,chanMask=obs.calibration.phot.chanMask)
```

In the following example, we process a SDT using both the channel mask and the channel number mapping. The latter will be used to remove disconnected channels.

```
 IA>> sdt=maskBadChan(sdt,chanMask=obs.calibration.spec.chanMask,
chanNum=obs.calibration.spec.chanNum)
```

Note that if you use a spectrometer calibration product on a photometer detector timeline (or viceversa) the task will throw a `SignatureException`

3.5.8. Error messages

TBW. TBW.

TBW. TBW.

3.6. The *Engineering Conversion* Pipeline Step-by-Step: Time Conversion

3.6.1. Module Description

The *Time Conversion* Module implements the time conversion and reordering.

3.6.2. Input Data Products

rawData	Input timeline to be processed.

3.6.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.6.4. Input Calibration Products

resetHist	DPU reset history calibration product.

3.6.5. How to use the `TimeConvReordTask`

The `TimeConvReordTask` implements the time conversion and reordering.

3.6.6. Parameter Options

Users can set the following parameters of the module:

3.6.7. Examples

In the following example, we process a PDT.

```
 IA>> obs=.... #obs is the ObservationContext
```

```
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of building
block 0xA0300001
 IA>>
 IA>> pdt=formatConversion(rpdt) # reformatting
 IA>> pdt=detFlagger(pdt)
 IA>> pdt=maskBadChan(pdt,chanMask=obs.calibration.phot.chanMask)
 IA>> pdt=timeConvReord(pdt,resetHist=obs.calibration.resetHist)
```

In the following example, we process a Raw Nominal Housekeeping Timeline. Note that in this case we don't need a Reset History calibration product.

```
 IA>> obs=.... #obs is the ObservationContext
 IA>> obsid=obs.obsid
 IA>> bbid=0xA0300001L
 IA>> rpdt=obs.level["level0"].get(obsid,bbid).rnhkt # extract the RNHKT of
building block 0xA0300001
 IA>>
 IA>> nhkt=formatConversion(rnhkt) # reformatting
 IA>> nhkt=timeConvReord(nhkt)
```

3.6.8. Error messages

TBW. TBW.

TBW. TBW.

3.7. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the JFET Voltages

3.7.1. Module Description

The *Calculate JFET Voltages* Module implements the conversion of detector signals from ADUs to JFET voltages.

3.7.2. Input Data Products

rawData	Input detector timeline to be processed.
nhkt	Nominal Housekeeping Timeline. If the NHKT is provided, the bias frequency is taken from this product and not from the metadata of the detector timeline (if it was set).

3.7.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.7.4. Input Calibration Products

offset	Detector offset history calibration product.
chanGain	Channel gains calibration product. This parameter is mandatory.

3.7.5. How to use the *Calculate JFET Voltages* Module

The `CalcJfetVoltTask` implements the conversion of detector signals from ADUs to JFET voltages.

3.7.6. Parameter Options

biasFreq — The bias frequency in Hz. If this value is set, the `nhkt` parameter is ignored as well as the bias frequency value in the input detector timeline metadata.

3.7.7. Examples

In the following example, we process a Photometer Detector Timeline, providing the offset history, the channel gain and the housekeeping. In this case, the voltage will be computed using equation (2) of the SPIRE Pipeline Description document. The bias frequency will be obtained from the Nominal Housekeeping Timeline.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>>
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain, nhkt=nhkt)
```

In the following example, we process a Photometer Detector Timeline, providing the offset history, the channel gain and a bias frequency. As before, the voltage will be computed using equation (2) of the SPIRE Pipeline Description document.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>>
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain, biasFreq=131.7)
```

As the previous example, but setting the bias frequency in the input PDT.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>> pdt.biasFreq=131.7
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain)
```

As the previous example, but without providing the offset history. In this case the voltage will be computed assuming an offset of 0 for all channels.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> chanGain=... #the channel gain table
IA>> pdt.biasFreq=131.7
IA>> pdt=calcJfetVolt(pdt, chanGain=chanGain)
```

3.7.8. Error messages

TBW. TBW.

TBW. TBW.

3.8. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the bolometer RMS voltage and resistance

3.8.1. Module Description

The *Calculate Bolometer RMS Voltage* Module implements the calculation of bolometer RMS voltage and resistance from the JFET voltages.

3.8.2. Input Data Products

data	Input detector timeline to be processed.
nhkt	Nominal Housekeeping Timeline. The NHKT is needed to get the bias amplitudes. It is a mandatory input.

3.8.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.8.4. Input Calibration Products

chanNum	Channel number mapping calibration product.
chanGain	Channel gains calibration product. This parameter is mandatory.
balPar	Bolometer parameters calibration product.
chanNomRes	Channel nominal resistances calibration product.

3.8.5. How to use the *Calculate JFET Voltages* Module

The `CalcRmsVoltResTask` implements the calculation of RMS bolometers voltage and resistance from JFET voltages.

3.8.6. Parameter Options

Users can set the following parameters of the module:

3.8.7. Examples

In the following example, we process a Photometer Detector Timeline, providing only the channel gain and the housekeeping. In this case only the voltages can be computed; this will be done with the equation $V_{d\text{-}RMS}=V_{JFET\text{-}RMS}/H_{JFET}$.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
```

```
 IA>> chanGain=... #the channel gain table
 IA>> nhkt=... #  Nominal Housekeeping Timeline
 IA>>
 IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt)
```

In the following example, we also provide the bolometer parameters table. Voltages and resistances will be computed using equations in section 3.8 of the SPIRE Pipeline Description document; but assuming a phase shift $\Delta\varphi=0$. Since the channel number mapping is not provided, the tasks assumes that thermal control channels are those with a name starting as "PTC" or "PMWP". For thermal control channels, the resistance is set to NaN, and the voltage is simply $V_{d\text{-}RMS}=V_{JFET\text{-}RMS}/H_{JFET}$.

```
 IA>> pdt=... #pdt is a Photometer Detector Timeline
 IA>> nhkt=... #  Nominal Housekeeping Timeline
 IA>> chanGain=... #the channel gain table
 IA>> bolPar=... #the bolometer parameters table
 IA>>
 IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt, bolPar=bolPar)
```

In the following example, we also provide the channel nominal resistances and the channel number mapping product. Voltages and resistances and phase shifts will be computed using equations in section 3.8 of the SPIRE Pipeline Description document. Thermal control channels names will be taken form the channel number mapping product.

```
 IA>> pdt=... #pdt is a Photometer Detector Timeline
 IA>> nhkt=... #  Nominal Housekeeping Timeline
 IA>> chanGain=... #the channel gain table
 IA>> bolPar=... #the bolometer parameters table
 IA>> chanNomRes=... #the channel nominal resistances table
 IA>> chanNum=... #the channel number mapping table
 IA>>
 IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt, bolPar=bolPar,
chanNum=chanNum, chanNomRes=chanNomRes)
```

3.8.8. Error messages

TBW. TBW.

TBW. TBW.

3.9. The *Engineering Conversion* Pipeline Step-by-Step: Adding Pointing Meta-Data Information to the Data

3.9.1. Module Description

The *Add Pointing Meta-Data* Module implements the computation of pointing metadata parameters.

3.9.2. Input Data Products

data	Input detector timeline to be processed.

nhkt Nominal Housekeeping Timeline.

3.9.3. Output Data Products

Level 0.5 style products **Level 0.5 style products** .

3.9.4. Input Calibration Products

No calibration data are needed.

3.9.5. How to use the *Add Pointing Meta-Data* Module

The *Add Pointing Meta-Data* Module implements the computation of pointing metadata parameters by the `AddPointingParamTask` task.

3.9.6. Input Control Parameters

step The STEP value. If this value is set, the `nhkt` is ignored.

3.9.7. Examples

In the following example, we process a Photometer Detector Timeline, providing the housekeeping.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>>
IA>> pdt=addPointingParam(pdt, nhkt=nhkt)
```

In the following example, we provide directly the STEP value.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>>
IA>> pdt=addPointingParam(pdt, step=12345)
```

3.9.8. Error messages

TBW. TBW.

TBW. TBW.

Chapter 4. Processing Data with the Photometer Large Scan Map Pipeline

4.1. Introduction

A detailed description of the photometer pipeline design can be found in

Griffin M., Dowell D., Lim T., Bendo G., Bock J., Cara C., Castro-Rodriguez N., Chanial P., Clements D., Gastaud R., Guest S., Glenn J., Hristov V., King K., Laurent G., Lu N., Mainetti G., Morris H., Nguyen H., Panuzzo P., Pearson C., Pinsard F., Pohlen M., Polehampton E., Rizzo D., Schulz B., Schwartz A., Sibthorpe B., Swinyard B., Xu K., Zhang L., The Herschel-SPIRE Photometer Data Processing Pipeline, Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

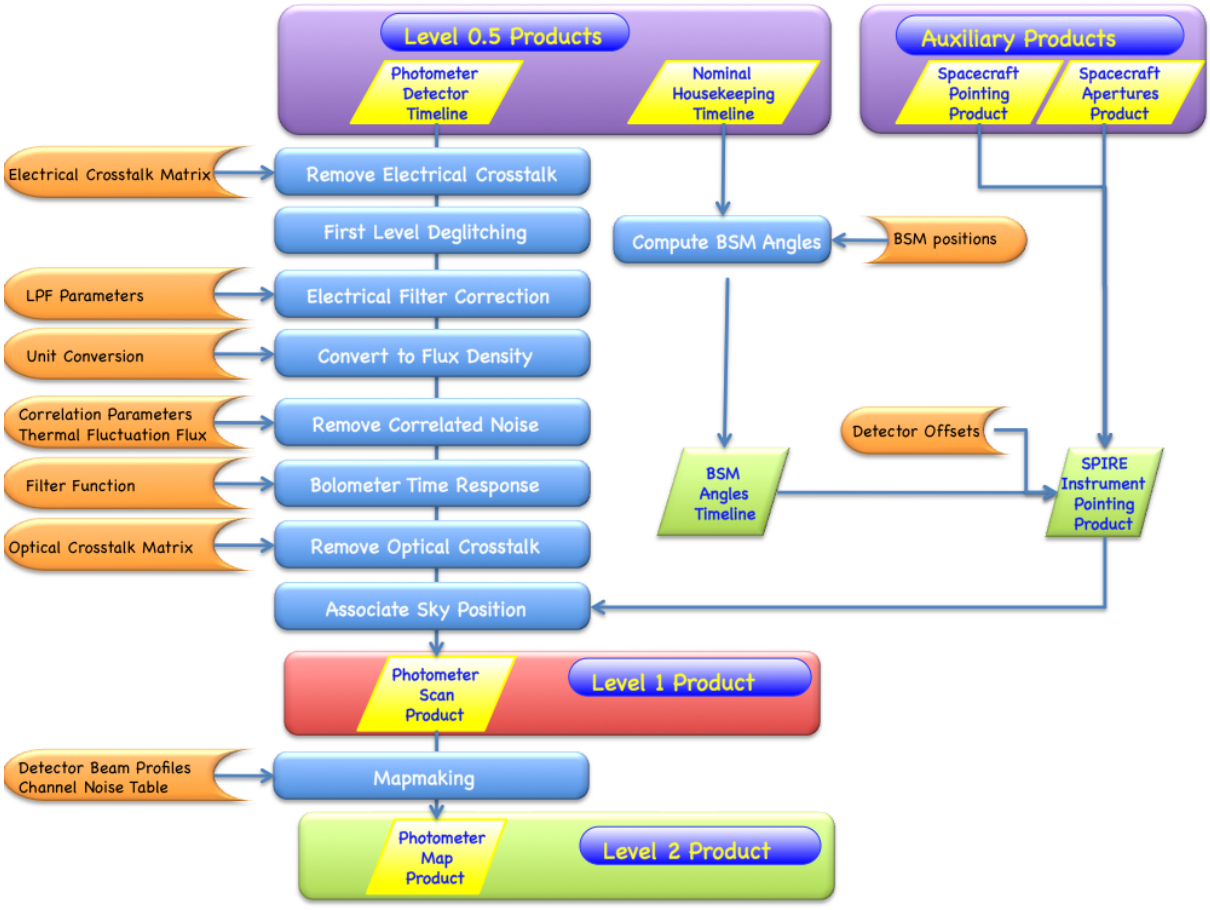

Figure 4.1. Flowchart for the Photometer Scan Map Pipeline.

4.2. Conversion of the BSM telemetry into Angles on the Sky

4.2.1. Module Description

The *Compute BSM Angles* Module converts the angles of the Beam Steering Mirror to physical units (decimal degree in the sky). The method is simple: interpolation from a calibration table containing the two converted angles (Y,Z) versus the two raw angles (chopper angle, jiggle angle). Note that for scan map observations there is no chopping and therefore, there is no need to create a chop-jiggle timeline as in the case of jiggle observations; however it is still necessary to create a BSM angle timeline because it will be necessary to know where the BSM is pointing, even if it should be at a fixed position. However, for scan map observations, the instrument does not produce BSM telemetry packets at all, so instead the input comes from the Nominal Housekeeping Timeline (NHK) which contains the BSM sensor values. This process extracts focal plane Y,Z angles corresponding to the sample time in the NHK timeline by comparing the sensor signals in the NHK timeline and BSM Positions Table which contains the Y,Z angles for a given chop and jiggle BSM sensor value

4.2.2. Input Data Products

NHKT	**Nominal House Keeping Timeline** This contains the timeline of the positions of the two mirrors as sensor signal values (the chopper and the jiggle mirror), in raw format in the `chopsenssig` and `jiggsenssig` columns.

4.2.3. Output Data Products

BAT	**BSM Angles Timeline** It contains timelines of angular distance on the sky from its zero position (in spacecraft Y, Z coordinates . For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

4.2.4. Input Calibration Products

SCalPhotBsmPos'	**BSM Position Table** This Table contains BSM sensor signal in raw units (in chop and jiggle directions) and angular distance on the sky from its zero position (in spacecraft Y, Z coordinates). There is one table for the photometer and another for the spectrometer, because the reference bsm position can be different. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

4.2.5. How to use the *Compute BSM Angles* Module

The module takes a `EdpProduct` object as input. This contains the timeline of the two angles of the mirror (chop and jiggle) in raw data contained in the Nominal House Keeping Timeline (NHKT).

The module also needs a calibration product `BsmPos`. It contains information for converting the raw angles.

The task output is a `BSM Angles Timeline` which contains the timeline of the angles Y and Z (BAT).

4.2.6. Parameter Options

Users can set the following parameters of the module:

4.2.7. Examples

Assuming that `nhkt`, a NHKT product, and `bsmPos` is a BsmPos product:

```
bat=calcBsmAngles(nhkt,bsmPos)
yang=bat.yangle
zang=bat.zangle
t0=bat.sampleTime[0]
time=bat.sampleTime-t0
p2=PlotXY(time,yang,xtitle="Time (sec)",ytitle="Chopper/Jiggle angle (arcsec)",
 titleText="BSM angle timeline ")
p2[0].name="Chopper"
p2[1]=LayerXY(time,zang,name="Jiggle")
p2.legend.visible=1

p3=PlotXY(yang,zang,xtitle="Chopper angle (arcsec)",ytitle="Jiggle angle (arcsec)",
 titleText="BSM positions ",line=0,symbol=3)
p3[0].name="BAT"
```

The process is shown pictorially in the figure below. Note that the interpolation method is not a parameter for the moment.

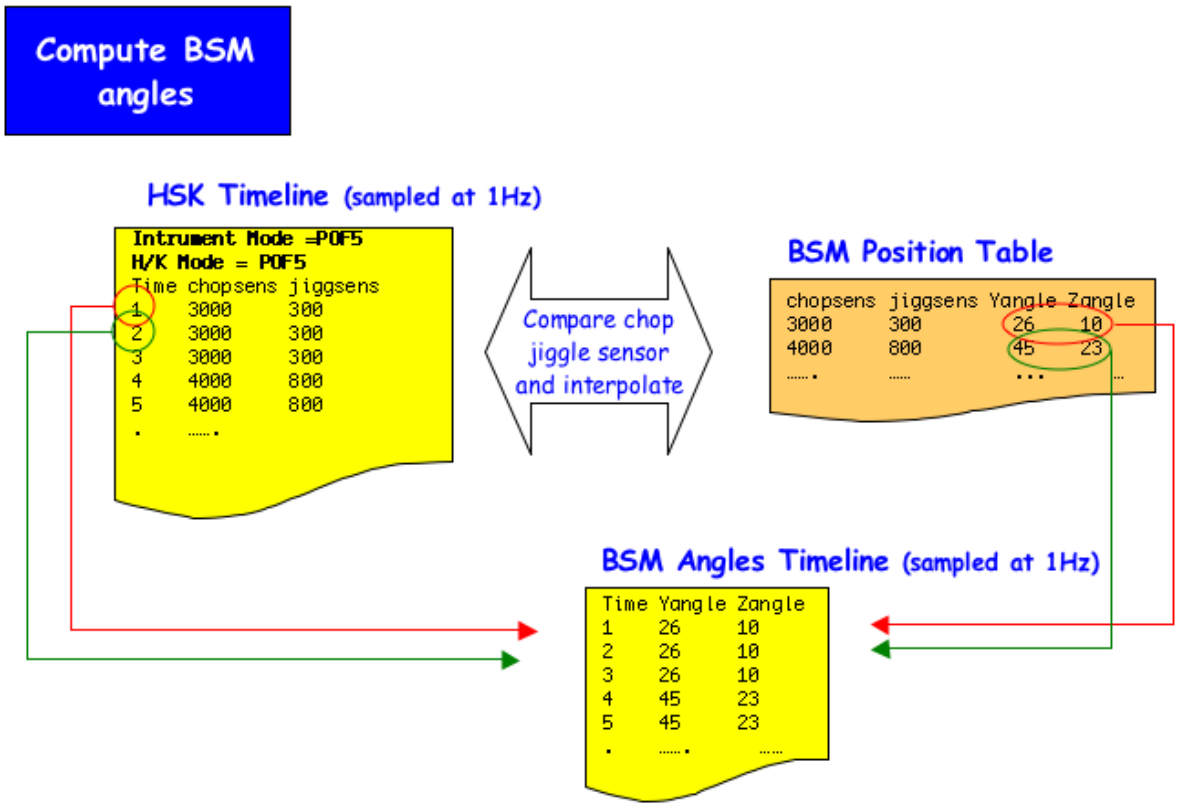

Figure 4.2. Creation of BSM Angles Timeline by the Compute BSM Angles module in the Scan Map Pipeline.

4.2.8. Error messages

TBW. TBW.

TBW. TBW.

4.3. Deglitching the Timeline Data

4.3.1. Module Description

The *First Level Deglitcing* Module removes glitches due to cosmic ray hits or other impulse-like events. The basic assumption is that the glitch signature is similar to a Dirac delta function. This process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs a signal free of such glitch signatures.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

4.3.2. Input Data Products

PDT; **Photometer Detector Timeline** The input PDT product contains timelines for each photometer detector for each observation building block.

4.3.3. Output Data Products

PDT; **Spectrometer Detector Timeline** The output PDT product contains deglitched timelines for each spectrometer detector for each observation building block.

4.3.4. Input Calibration Products

No calibration data are needed.

4.3.5. How to use the *First Level Deglitching* Module

4.3.6. Parameter Options

The first level deglitching task employs a complex algorithm, based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent complex processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the extended documentation for this task.

scaleMin, scaleMax, and voices — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMaxe. The current best estimation for the Scanmap Pipeline are values of scaleMin = 1, scaleMax= 8 and voices = 5.. voices defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the number of voices per range unit is directly proportional to execution time. Both should therefore be kept at the minimum required to accurately determine the Holder index.

thresholdHolder and hMin — thresholdHolder/hmin are the minimum/maximum values of a real data type of the range of slopes which are interpreted as glitch indica-

tors. The range should include -1 and thresholdHolder should be larger than hmin. The current best estimation for the Scanmap Pipeline are values of thresholdHolder = -0.3 and hMin = -1.9. The range defined by thresholdHolder and hmin allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

thresholdCorr — This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This real number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. The current best estimation for the Scanmap Pipeline are values of thresholdCorr = 0.7. The value selected for thresholdCorr allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

4.3.7. Examples

```
IA>> from herschel.spire.ia.pipeline.common.deglitch import DeglitchingTask
IA>>
IA>> deglitch=DeglitchingTask()
IA>> pdt=deglitch(pdt,
                      scaleMin=1,
     scaleMax=8,
     voices=5,
     hMin=-1.9,
     thresholdHolder=-0.3,
     thresholdCorr=0.7)
IA>>
```

4.3.8. Error messages

severe :unable to store output product. The task failed to write the output product.

warning :Unable to perform reconstruction. The task failed to reconstruct signal samples.

severe :unable to perform wavelet decomposition.. The task failed to identify glitches because the signal could not be decomposed into its wavelet components.

4.4. Removing Electrical Crosstalk from the Timeline Data

4.4.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Detector Timelines. The module multiplies a crosstalk removal matrix with the signal vector for each time sample and returns Detector Timelines which are corrected for electrical crosstalk.

4.4.2. Input Data Products

PDT **Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.4.3. Output Data Products

PDT **Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical crosstalk.

4.4.4. Input Calibration Products

SCalElecCross **Electrical Crosstalk Matrix** There are three electrical crosstalk matrices one for each of the SPIRE photometer arays. They contain entries for the electrical crosstalk between the detectors of each one of the two detector arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

4.4.5. How to use the *Remove Electrical Crosstalk* Module

It should be noted that, prior to launch, no electrical crosstalk was observed. During the fabrication of the warm read-out electronics, spot-checking showed extremely low levels of electrical crosstalk. An additional analysis was performed for the detector arrays of the SPIRE imaging photometer using data from the ground-based test campaigns. This analysis checked whether the random instances of detectors absorbing ionizing radiation lead to reproducible signals in other detectors. Again, this analysis has shown no measurable evidence for electrical crosstalk. Close examination of the post-launch performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this module simply multiplies the detector timelines by the identity matrix.

4.4.6. Parameter Options

N/A. There are no options for this module.

4.4.7. Examples

4.4.8. Error messages

TBW. TBW.

TBW. TBW.

4.5. Correcting for Electrical Filter Response

4.5.1. Module Description

The *Correction for Electrical Filter Response* module is responsible for removing the effects of the electrical filter (Low pass filter) transient response. This is achieved by:

1. Transforming the signal timelines of individual detectors in to the Fourier domain
2. Dividing by the electrical filter transfer function (described in AD05, section 3.5.3)
3. Transforming the result back to the time domain

4.5.2. Input Data Products

PDT	**Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.5.3. Output Data Products

PDT	**Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical filter response.

4.5.4. Input Calibration Products

SCalPhotElecFiltCorr	**Electrical Filter Correction** The calibration product provides information on the parameters describing the bolometer transfer function (AD01, section 4.1.5).

4.5.5. How to use the *Correction for Electrical Filter Response* Module

4.5.6. Parameter Options

N/A. There are no options for this module.

4.5.7. Examples

```
                              IA>> from
herschel.spire.ia.pipeline.phot.elecfiltcorr import CorrElecFiltResponseTask
                              IA>>
                              IA>>
Mytask=CorrElecFiltResponseTask.CorrElecFiltResponseTask()
                              IA>>
                              IA>> PDT_IN=Mytask(PDT_IN) # where PDT_IN is the
input Phot. Det. Timeline product
                              IA>> #The above command will correct the timelines
included in the input product
                              IA>> #for the effects of the electrical filter.
```

4.5.8. Error messages

TBW. TBW.

TBW. TBW.

4.6. Converting the Detector Timelines into Flux Units

4.6.1. Module Description

The *Flux Conversion* Module applies a correction for the nonlinear bolometer responsivity to a photometer detector voltage time line V (i.e., PDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector channel in a calibration product (`SCalPhotFluxConv()`. The result is an in-beam flux density timeline.

4.6.2. Input Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal in electrical units (i.e. Volts)

4.6.3. Output Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) product with bolometer signal in units of Jy

4.6.4. Input Calibration Products

ScalPhotFLuxConv	**Photometer Flux Conversion and Non-linearity Correction Coefficients** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the calibrated limiting bolometer voltages.

4.6.5. How to use the *Flux Conversion* Module

4.6.6. Parameter Options

There are no parameter options for this module.

4.6.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data processed to Level 0.5, with bolometer signal data in units of volts `PDT` and a calibration product, `phot-fluxconv`.

```
IA>> detectortimeline = PDT
IA>> calibrationproduct = phot-fluxconv
IA>> task = FluxConversionTask()
IA>> outputfluxproduct = task(detectortimeline,calibrationproduct)
IA>>
```

4.6.8. Error messages

Error messages generated:

If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

4.7. Removing Correlated Noise from the Detector Timelines

4.7.1. Module Description

The *Remove Correlated Noise* Module subtracts low frequency (< 1Hz) noise, caused by variations of the detector array bath temperature, from scan mapping data time lines.The module is based on the empirical correlations between detectors and thermistors (or, in case of high bias voltages when thermistors are saturated, the correlations between detectors and dark pixels). The module shall be run after the correction for any nonlinearity (the flux conversion module).

4.7.2. Input Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal converted to flux units (e.g. Jy.) by Photometer Flux Conversion Module.

4.7.3. Output Data Products

PDT	**Photometer Detector Timeline** Data timelines in the units of Jy, corrected for temperature drift.

4.7.4. Input Calibration Products

ScalPhotTempDriftCorr	**Temperature Drift Correction Parameter Table** A product containing for each detector array, a thermistor selection flag, the base voltage values for each thermistor, the errors for those values, and a validity calibration flag for those values. It contains for each bolometer channel, correction coefficients A1, A1 error, B1, B1 error, AB1 flag (all for thermistor 1) and similar values for thermistor 2.

4.7.5. How to use the *Remove Correlated Noise* Module

4.7.6. Parameter Options

Users can set the following parameters of the module:

timeSpan	the timeSpan (in sec) used for averaging the thermistor timeline prior to performing spline function smoothing.

4.7.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data, `PDT` processed by the Photometer Flux Conversion Module with bolometer signal data in units of Janskys. A tempdrift calibration product, `tempDrift`.

```
IA>> detectortimeline = PDT
IA>> calibrationproduct = tempDrift
IA>> task = TemperatureDriftCorrectionTask()
IA>>  timeSpan = 5  # this is the smoothing timespan in seconds.
```

```
IA>> outputfluxproduct = task(detectortimeline,calibrationproduct,timeSpan)
IA>>
```

4.7.8. Error messages

TBW. None

TBW. None

4.8. Correcting for the Bolometer Time Response

4.8.1. Module Description

The *Bolometer Time Response Correction* Module is responsible for removing the effects of the bolometer transient response. This is achieved by:

1. Transforming the signal timelines of individual detectors in to the Fourier domain
2. Dividing by the bolometer transient response transfer function (described in AD05, section 4)
3. Transforming the result back to the time domain

4.8.2. Input Data Products

PDT	**Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.8.3. Output Data Products

PDT	**Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for bolometer time response.

4.8.4. Input Calibration Products

ScalPhotDetTimeConst	**Photometer Detector Time Constants** Information on the parameters describing the bolometer transfer function will be provided by the(ScalPhotDetTimeConst) calibration product (AD01, section 4.1.8).

4.8.5. How to use the *Bolometer Time Response Correction* Module

4.8.6. Parameter Options

Users can set the following parameters of the module:

4.8.7. Examples

```
                                IA>> from
herschel.spire.ia.pipeline.phot.bolorespcorr import CorrBolTimeResponseTask
                                IA>>
                                IA>>
Mytask=CorrBolTimeResponseTask.CorrBolTimeResponseTask()
                                IA>>
                                IA>> PDT_IN=Mytask(PDT_IN) # where PDT_IN is the
input Phot. Det. Timeline product
                                IA>> #The above command will correct the timelines
included in the input product
                                IA>> #for the effects of the bolometer transient
response.
```

4.8.8. Error messages

TBW. TBW.

TBW. TBW.

4.9. Removing the Effects of Optical Crosstalk from the Data

4.9.1. Module Description

The *Optical Crosstalk Removal* Module

4.9.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

4.9.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

4.9.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

4.9.5. How to use the *Optical Crosstalk Removal* Module

4.9.6. Parameter Options

Users can set the following parameters of the module:

4.9.7. Examples

4.9.8. Error messages

TBW. TBW.

TBW. TBW.

4.10. Adding Positional Information (WCS) to the Data

4.10.1. Module Description

The *Associate Sky Position* Module

4.10.2. Input Data Products

Input Product name here	**Input Product name here** and description to follow here

4.10.3. Output Data Products

Output Product Here	**Output Product Name Here** and description to follow here

4.10.4. Input Calibration Products

Calibration Product Name Here	**Calibration Product Name Here** and description to follow here

4.10.5. How to use the *Associate Sky Position* Module

4.10.6. Parameter Options

Users can set the following parameters of the module:

4.10.7. Examples

4.10.8. Error messages

TBW. TBW.

TBW. TBW.

4.11. Converting the On-Board Time to International Time Standards

4.11.1. Module Description

The *Time Correction* Module

4.11.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

4.11.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

4.11.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

4.11.5. How to use the *Time Correction* Module

4.11.6. Parameter Options

Users can set the following parameters of the module:

4.11.7. Examples

4.11.8. Error messages

TBW. TBW.

TBW. TBW.

4.12. Creating Image Scan-Maps

4.12.1. Module Description

The *Map Making* provides the map-making routines used by the SPIRE photometer pipeline. Scan mode timelines are combined to create a map using a choice of two algorithms: direct coaddition (`NaiveScanMapperTask`) and maximum-likelihood estimate (`MadScanMapperTask`). All output maps are North-oriented.

4.12.2. Input Observational Data Products

PSP

Photometer Scan Product The `NaiveScanMapperTask` and `MadScanMapperTask` take one Photometer Scan Product (PSP) or a context of PSP as input. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

4.12.3. Output Data Products

All map-making tasks returns the map associated to the bolometer array specified by the user. The map is stored as a `SimpleImage`. For a definition, see:

ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/api/herschel/ia/dataset/image/SimpleImage.html

4.12.4. Input Calibration Products Required

The `MadScanMapperTask` requires a Dtector Noise Spectrum for thephotometer channel noise. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

For the `NaiveApppMapperTask` there are no calibration files required for the Jiggle Map `NaiveApppMapperTask`
.

4.12.5. How to use the *Map Making* Module

The module contains 3 mappers: 2 for the scan mode (`NaiveScanMapperTask` and `MadScanMapperTask`) and 1 for the jiggle mode (`NaiveApppMapperTask`).

The input parameters are a context of Photometer Detector Timelines (`PointedProduct`) for the scan mode and an Averaged PointedPhotometer Product for the jiggle mode.

The user should set the `array` keyword to "PSW", "PMW" or "PLW" to select the bolometer array to be processed. The optional keyword `maxmemory` can be used to increase the memory to be allocated for map-making and avoid slowing down by limiting I/O. The termination criteria for the task `MadScanMapperTask` can also be modified: the keywords `maxiterationpcg` and `maxrelerrorpcg` specify the maximum number of iterations and the maximum allowable residual in the Preconditionned Conjugate Gradient method.

The mappers only compute one map at a time, for the bolometer array specifed by the user. The output map is stored as a `SimpleImage` product, for which convenient visualisation tools are available (see examples below) within HCSS.

The mappers use temporary files to store the image coordinates (X,Y). `IOException` will be thrown is these temporary files can not be written or read. Out of memory exception can occur if the size of the map is too large (because of limited resource or incorrect astrometry)

4.12.6. Parameter options

Users can set the following parameters of the Map Making module:

array

Name of the bolometer array to be processed The value of this parameter can be "PSW", "PMW" or "PLW". If not set, the default bolometer array is assumed to be "PSW".

resolution	**Pixel size of the output map** The value of this parameter is in arcseconds. If not set, the default values are 6", 10" and 14 for the PSW, PMW and PLW bolometer array.
maxmemory	**Maximum memory in bytes to be used to hold the pointing matrix and the timeline in memory.** If the allocated memory isn't large enough, processing time may be I/O limited. By default, 1GB is assumed.
maxiterationpcg	**The maximum number of Preconditionned Conjugate Gradient (PCG) iterations before termination** (`MadScanMapperTask` only). By default, the maximum number of iterations is 50.
maxrelerrorpcg	**The largest allowable relative residual in PCG** (`MadScanMapperTask` only). By default, its value is 1e-6.

4.12.7. Examples

For scan mode observations, the user has to provide a set of building blocks, organised as a `Context` of level-1 `PointedProduct`.

Here is an example in which the PointedProduct Context is obtained from the Observation Context

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import NaiveScanMapperTask
IA>> mapper = NaiveScanMapperTask()
IA>> scanCon=obs.level1.getScan()
IA>> psw1=mapper(scanCon, array='PSW')
IA>> Display(psw1)
```

To use the maximum-likelihood mapper (MADMap), the synthax is similar.

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import MadScanMapperTask
IA>> mapper = MadScanMapperTask()
IA>> psw2 = mapper(scanCon, array='PSW')
IA>> Display(psw2)
```

4.12.8. Error messages

4.13. Point Source Extraction

4.13.1. Module Description

The *Point Source Extraction* Module

4.13.2. Input Observational Data Products

4.13.3. Output Data Products

4.13.4. Input Calibration Products Required

4.13.5. How to use the *Point Source Extraction* Module

4.13.6. Parameter options

Users can set the following parameters of the Demodulation module:

4.13.7. Examples

4.13.8. Error messages

Chapter 5. Processing Data with the Photometer Jiggle Map Pipeline

5.1. Introduction

A detailed description of the photometer pipeline design can be found in

Griffin M., Dowell D., Lim T., Bendo G., Bock J., Cara C., Castro-Rodriguez N., Chanial P., Clements D., Gastaud R., Guest S., Glenn J., Hristov V., King K., Laurent G., Lu N., Mainetti G., Morris H., Nguyen H., Panuzzo P., Pearson C., Pinsard F., Pohlen M., Polehampton E., Rizzo D., Schulz B., Schwartz A., Sibthorpe B., Swinyard B., Xu K., Zhang L., The Herschel-SPIRE Photometer Data Processing Pipeline, Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

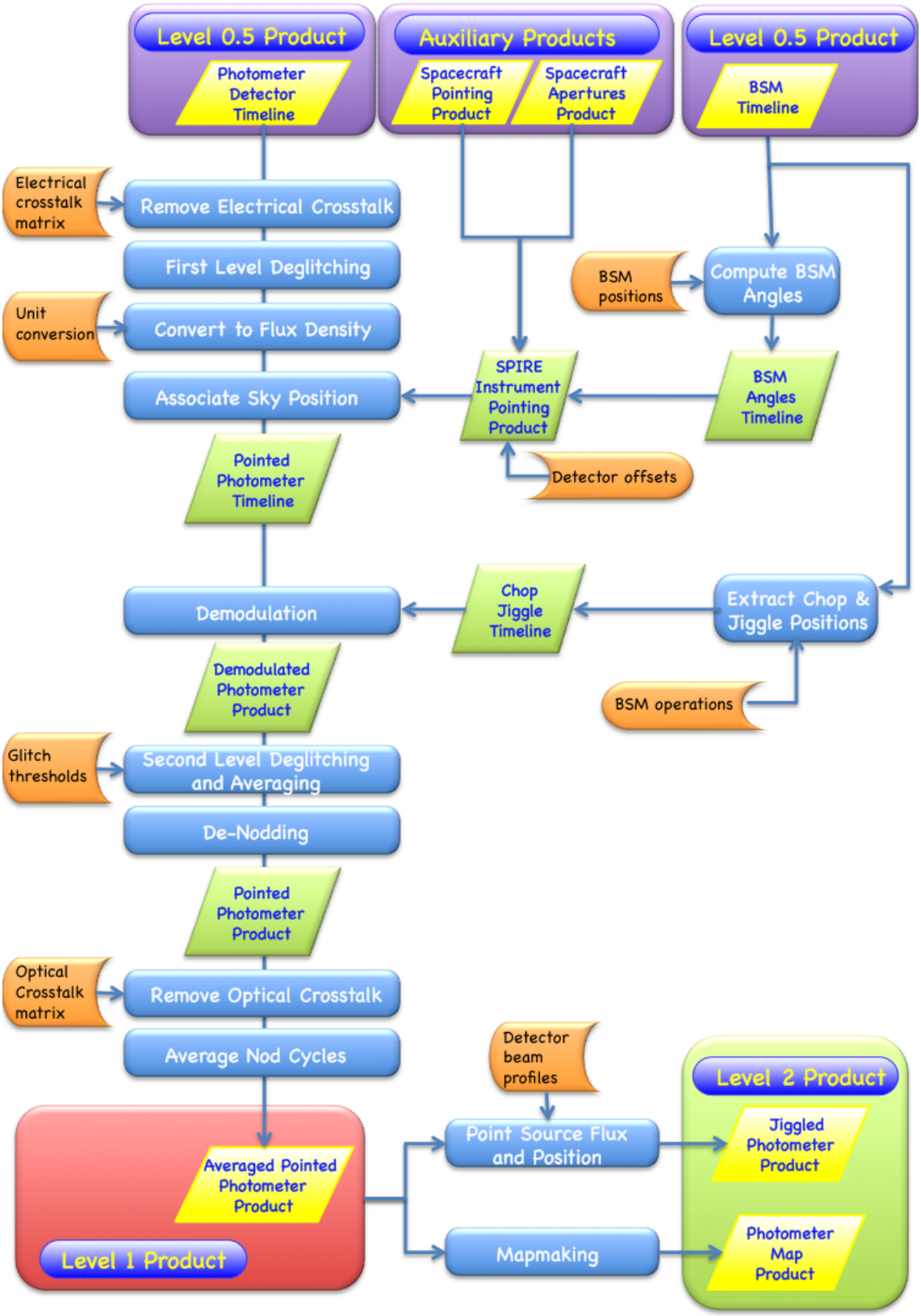

Figure 5.1. Flowchart for the Photometer Jiggle Map Pipeline.

5.2. Conversion of the BSM telemetry into Angles on the Sky

5.2.1. Module Description

The *Compute BSM Angles* Module converts the angles of the Beam Steering Mirror to physical units (decimal degree in the sky). The method is simple: interpolation from a calibration table containing the two converted angles (Y,Z) versus the two raw angles (chopper angle, jiggle angle).

5.2.2. Input Data Products

BSMT	**Beam Steering Mirror Timeline** This contains the timeline of the positions of the two mirrors, the chopper and the jiggle mirror, in raw format. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.2.3. Output Data Products

BAT	**BSM Angles Timeline** It contains timelines of angular distance on the sky from its zero position (in spacecraft Y, Z coordinates . For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.2.4. Input Calibration Products

SCalPhotBsmPos'	**BSM Position Table** This Table contains BSM sensor signal in raw units (in chop and jiggle directions) and angular distance on the sky from its zero position (in spacecraft Y, Z coordinates). There is one table for the photometer and another for the spectrometer, because the reference bsm position can be different. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.2.5. How to use the *Compute BSM Angles* Module

The module takes a `EdpProduct` object as input. This contains the timeline of the two angles of the mirror (chop and jiggle) in raw data (BSMT).

The module also needs a calibration product `BsmPos`. It contains information for converting the raw angles.

The task output is a `BSM Angles Timeline` which contains the timeline of the angles Y and Z (BAT).

5.2.6. Parameter Options

Users can set the following parameters of the module:

5.2.7. Examples

Assuming that `bsmt` is a BSMT product, and `bsmPos` is a BsmPos product:

```
bat=calcBsmAngles(bsmt,bsmPos)
yang=bat.yangle
zang=bat.zangle
t0=bat.sampleTime[0]
time=bat.sampleTime-t0
p2=PlotXY(time,yang,xtitle="Time (sec)",ytitle="Chopper/Jiggle angle (arcsec)",
 titleText="BSM angle timeline ")
p2[0].name="Chopper"
p2[1]=LayerXY(time,zang,name="Jiggle")
p2.legend.visible=1

p3=PlotXY(yang,zang,xtitle="Chopper angle (arcsec)",ytitle="Jiggle angle (arcsec)",
 titleText="BSM positions ",line=0,symbol=3)
p3[0].name="BAT"
```

Note that the interpolation method is not a parameter for the moment.

5.2.8. Error messages

TBW. TBW.

TBW. TBW.

5.3. Extracting the Chop and Jiggle Positions from the BSM Timeline

5.3.1. Module Description

The *Extract Chop and Jiggle Positions* Module extracts from the raw values of the angles of the Beam Steering Mirror the position of the jiggle map and the chopper information. The BSM has a definite position, then move to another one. We have a start time and an end time for each position. The method is simple: each couple (chopper angle, jiggle angle) is compared to each position of the Operation calibration table with a tolerance. If it falls inside the rectangle defined by the tolerance, it is in this position, see figure below

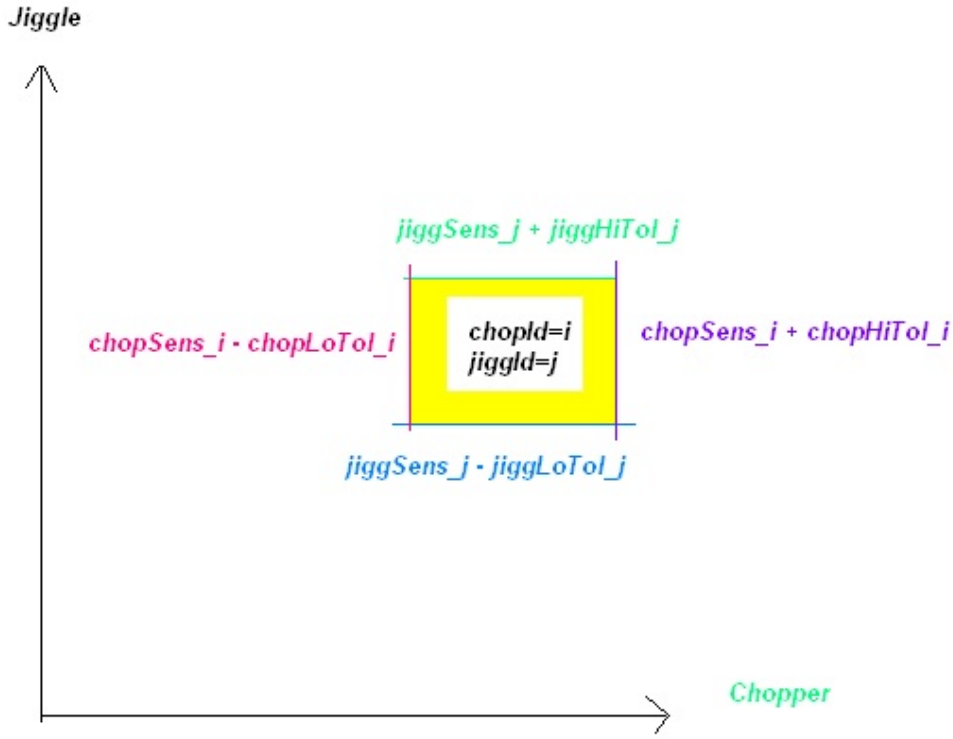

Figure 5.2. BSM Operation Table

5.3.2. Input Data Products

BSMT — **Beam Steering Mirror Timeline** This contains the timeline of the positions of the two mirrors, the chopper and the jiggle mirror, in raw format. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.3.3. Output Data Products

CJT — **Chop Jiggle Timeline** The final output of this module is Chopper Jiggle Timeline (CJT). It contains the start time and end time and the id of each consecutive jiggle map position. In addition, for debugging purpose, the converted values Y,Z of each jiggle map position is also stored. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.3.4. Input Calibration Products

SCalPhotBsmOps — **BSM Operations Table** This Table contains BSM sensor signal in raw units (in chop and jiggle directions) for each point of each jiggle map (remark: a 7 points jiggle map has 14 points because of the two chopper positions, On/Off). There is one table for the photometer and another for the spectrometer, because the reference bsm position can be different. For the product definition see: >http://www.spire.rl.ac.uk/icc/product_definitions/

5.3.5. How to use the *Extract Chop and Jiggle Positions* Module

The module takes a `EdpProduct` object as input. This contains the timeline of the two angles of the mirror (chop and jiggle) in raw data (BSMT).

The module needs two calibration products `BsmPos` and `BsmOps`.The first contains information for converting the raw angles, and the second the positions of the jiggle map.

The task output is a `Chop Jiggle Timeline` which contains the start time, end time, and id of each position in the jiggle map (CJT).

5.3.6. Parameter Options

Users can set the following parameters of the module:

5.3.7. Examples

Assuming that `bsmt` is a BSMT product, `bsmOps` is BsmOps product, and `bsmPos` is a BsmPos product. products:

```
calcBsmFlags = calcBsmFlags()
cjt = calcBsmFlags(bsmt=bsmt, bsmOps=bsmOps, bsmPos=bsmPos)
```

Note that CJT contains converted angles for debugging. This should vanish, and so the BsmPos calibration product should not be needed anymore.

5.3.8. Error messages

TBW. TBW.

TBW. TBW.

5.4. Deglitching the Timeline Data

5.4.1. Module Description

The *First Level Deglitcing* Module removes glitches due to cosmic ray hits or other impulse-like events. The basic assumption is that the glitch signature is similar to a Dirac delta function. This process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs a signal free of such glitch signatures.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

5.4.2. Input Data Products

PDT; **Photometer Detector Timeline** The input PDT product contains timelines for each photometer detector for each observation building block.

5.4.3. Output Data Products

PDT; **Spectrometer Detector Timeline** The output PDT product contains deglitched timelines for each spectrometer detector for each observation building block.

5.4.4. Input Calibration Products

No calibration data are needed.

5.4.5. How to use the *First Level Deglitching* Module

5.4.6. Parameter Options

The first level deglitching task employs a complex algorithm, based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent complex processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the extended documentation for this task.

scaleMin, scaleMax, and voices — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMaxe. The current best estimation for the Jiggle Pipeline are values of scaleMin = 1, scaleMax= 8 and voices = 5. voices defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the number of voices per range unit is directly proportional to execution time. Both should therefore be kept at the minimum required to accurately determine the Holder index.

thresholdHolder and hMin — thresholdHolder/hmin are the minimum/maximum values of a real data type of the range of slopes which are interpreted as glitch indicators. The range should include -1 and thresholdHolder should be larger than hmin. The current best estimation for the Jiggle Pipeline are values of thresholdHolder = -0.1 and hMin = -1.6. The range defined by thresholdHolder and hmin allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

thresholdCorr — This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This real number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. The current best estimation for the Jiggle Pipeline are values of thresholdCorr = 0.6. The value selected for thresholdCorr allows to

select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

5.4.7. Examples

```
IA>> from herschel.spire.ia.pipeline.common.deglitch import DeglitchingTask
IA>>
IA>> deglitch=DeglitchingTask()
IA>> pdt=deglitch(pdt,
                      scaleMin=1,
     scaleMax=8,
     voices=5,
     hMin=-1.6,
     thresholdHolder=-0.1,
     thresholdCorr=0.6)
IA>>
```

5.4.8. Error messages

severe :unable to store output product. The task failed to write the output product.

warning :Unable to perform reconstruction. The task failed to reconstruct signal samples.

severe :unable to perform wavelet decomposition.. The task failed to identify glitches because the signal could not be decomposed into its wavelet components.

5.5. Removing Electrical Crosstalk from the Timeline Data

5.5.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Detector Timelines. The module multiplies a crosstalk removal matrix with the signal vector for each time sample and returns Detector Timelines which are corrected for electrical crosstalk.

5.5.2. Input Data Products

PDT — **Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

5.5.3. Output Data Products

PDT — **Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical crosstalk.

5.5.4. Input Calibration Products

SCalElecCross — **Electrical Crosstalk Matrix** There are three electrical crosstalk matrices one for each of the SPIRE photometer arays. They contain entries for the electrical crosstalk

between the detectors of each one of the two detector arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

5.5.5. How to use the *Remove Electrical Crosstalk* Module

It should be noted that, prior to launch, no electrical crosstalk was observed. During the fabrication of the warm read-out electronics, spot-checking showed extremely low levels of electrical crosstalk. An additional analysis was performed for the detector arrays of the SPIRE imaging photometer using data from the ground-based test campaigns. This analysis checked whether the random instances of detectors absorbing ionizing radiation lead to reproducible signals in other detectors. Again, this analysis has shown no measurable evidence for electrical crosstalk. Close examination of the post-launch performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this module simply multiplies the detector timelines by the identity matrix.

5.5.6. Parameter Options

N/A. There are no options for this module.

5.5.7. Examples

5.5.8. Error messages

TBW. TBW.

TBW. TBW.

5.6. Converting the Detector Timelines into Flux Units

5.6.1. Module Description

The *Flux Conversion* Module applies a correction for the nonlinear bolometer responsivity to a photometer detector voltage time line V (i.e., PDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector channel in a calibration product (`SCalPhotFluxConv()`. The result is an in-beam flux density timeline.

5.6.2. Input Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal in electrical units (i.e. Volts)

5.6.3. Output Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) product with bolometer signal in units of Jy

5.6.4. Input Calibration Products

ScalPhotFLuxConv	**Photometer Flux Conversion and Non-linearity Correction Coefficients** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the calibrated limiting bolometer voltages.

5.6.5. How to use the *Flux Conversion* Module

5.6.6. Parameter Options

There are no parameter options for this module.

5.6.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data processed to Level 0.5, with bolometer signal data in units of volts `PDT` and a calibration product, `phot-fluxconv`.

```
IA>> detectortimeline = PDT
IA>> calibrationproduct = phot-fluxconv
IA>> task = FluxConversionTask()
IA>> outputfluxproduct = task(detectortimeline,calibrationproduct)
IA>>
```

5.6.8. Error messages

Error messages generated:

If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

5.7. De-Modulating the Timeline Data

5.7.1. Module Description

The *DeModulation* Module computes the amplitude of the modulation of the photometer dued to the motion of the chopper. There is one point per chopper cycle per bolometer.

5.7.2. Input Observational Data Products

PPP	**Pointed Photometer Product** A Pointed Photometer Product timeline as described in http://www.spire.rl.ac.uk/icc/product_definitions/
CJT	**Chop Jiggle Timeline** It contains the start time and end time and the id of each consecutive jiggle map position. In addition, for debugging purpose, the converted values Y,Z of each jiggle map position is also stored. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.7.3. Output Data Products

DPPP — **Demodulated Photometer Product** The module outputs a Demodulated Photometer Product desscribed by http://www.spire.rl.ac.uk/icc/product_definitions/.

5.7.4. Input Calibration Products Required

TBW — **not yet defined** The calibration product is not yet defined. It must contain two time contants, a delay time and a transition time. We don't know yet if these contants depend will vary for each bolometer or for each array of bolometers.

5.7.5. How to use the *Demodulation* Module

The Demodulation takes a `PointedPhotTimeline` object as input. This Pointed Photometer Product (PPP) contains the signal and astrometry timeline of each bolometer.

The Demodulation also takes a `CJT` object as input. This Chop Jiggle Product (CJT) contains the mirror angles timeline.

The Demodulation task output is `DemodPhotProduct` Photometer Demodulated Detector Product with one point per bolometer and per chopper cycle.

5.7.6. Parameter options

Users can set the following parameters of the Demodulation module:

5.7.7. Examples

Assuming that `ppt` is a Pointed Photometer Product and `cjt` is a CJT products:

```
demodulate=DemodulateTask()
dpp = demodulate(ppt=ppt, cjt=cjt)
```

Note that, since the calibration product for the time constants does not exist, default values are used.

5.7.8. Error messages

5.8. Averaging the Data of All Jiggle Positions (and Second Level Deglitching)

5.8.1. Module Description

The *Second Level Deglitching and Averaging* Module

5.8.2. Input Data Products

Input Product name here	**Input Product name here** and description to follow here

5.8.3. Output Data Products

Output Product Here	**Output Product Name Here** and description to follow here

5.8.4. Input Calibration Products

Calibration Product Name Here	**Calibration Product Name Here** and description to follow here

5.8.5. How to use the *Second Level Deglitching and Averaging* Module

5.8.6. Parameter Options

Users can set the following parameters of the module:

5.8.7. Examples

5.8.8. Error messages

TBW. TBW.

TBW. TBW.

5.9. De-Nodding Observations

5.9.1. Module Description

The *De-Nod* module de-nods data and averages multiple nodding cycles to return the signal per detector.

5.9.2. Input Observational Data Products

The module will take as input one, two or four Demodulated Photometer Products (DPP).

No product definition document available yet.

5.9.3. Output Data Products

The module outputs a Pointed Photometer Product (PPP).

See the Herschel Products Definition Document or alternatively http://www.spire.rl.ac.uk/icc/product_definitions/.

5.9.4. Input Calibration Products Required

No calibration products required.

5.9.5. How to use the *Denodding* Module

The module takes a `DenodInput` object as input. This is just a container for one, two or four Demodulated Photometer Products (DPP).

The module is made of two tasks, Denodding and Averaging. There is no wrapper task yet, so they will have to be called explicitly.

The Denodding task only takes one DenodInput as input and gives a Pointed Photometer Product (PPP) as output. The task must be executed for every set of nod positions (either A-B or A-B-B-A cycles).

The Averaging task takes an array of PPPs as input and gives a single PPP as output. Two optional parameters are available:

- **operator:** string parameter, can be either `mean` or `median`. Used to choose the averaging operator. Default is `mean`.
- **deglitch:** integer operator, used to toggle removal of outliers by setting it to either 0 or 1. Note that deglitching is not implemented at the moment. Default is 1.

5.9.6. Parameter options

Users can set the following parameters of the Denodding module:

5.9.7. Examples

Assuming that `dpp1` and `dpp2` are two DPP products:

```
denodInput = DenodInput()
denodInput.addProduct(dpp1)
denodInput.addProduct(dpp2)
denoddedPpp = DeNodding()(input=denodInput)
averagedPpp = Averaging()(input=[denoddedPpp])
```

Note that, since only one PPP is given as input to Averaging, the output product will contain the same values.

5.9.8. Error messages

5.10. Averaging the Data Taken from Different Nod Cycles

5.10.1. Module Description

The *Average Nod Cycles* Module

5.10.2. Input Data Products

Input Product name here	**Input Product name here** and description to follow here

5.10.3. Output Data Products

Output Product Here	**Output Product Name Here** and description to follow here

5.10.4. Input Calibration Products

Calibration Product Name Here	**Calibration Product Name Here** and description to follow here

5.10.5. How to use the *Average Nod Cycles* Module

5.10.6. Parameter Options

Users can set the following parameters of the module:

5.10.7. Examples

5.10.8. Error messages

TBW. TBW.

TBW. TBW.

5.11. Removing the Effects of Optical Crosstalk from the Data

5.11.1. Module Description

The *Optical Crosstalk Removal* Module

5.11.2. Input Data Products

Input Product name here	**Input Product name here** and description to follow here

5.11.3. Output Data Products

Output Product Here	**Output Product Name Here** and description to follow here

5.11.4. Input Calibration Products

Calibration Product Name Here	**Calibration Product Name Here** and description to follow here

5.11.5. How to use the *Optical Crosstalk Removal* Module

5.11.6. Parameter Options

Users can set the following parameters of the module:

5.11.7. Examples

5.11.8. Error messages

TBW. TBW.

TBW. TBW.

5.12. Converting the On-Board Time to International Time Standards

5.12.1. Module Description

The *Time Correction* Module

5.12.2. Input Data Products

Input Product name here	**Input Product name here** and description to follow here

5.12.3. Output Data Products

Output Product Here	**Output Product Name Here** and description to follow here

5.12.4. Input Calibration Products

Calibration Product Name Here	**Calibration Product Name Here** and description to follow here

5.12.5. How to use the *Time Correction* Module

5.12.6. Parameter Options

Users can set the following parameters of the module:

5.12.7. Examples

5.12.8. Error messages

TBW. TBW.

TBW. TBW.

5.13. Creating Image Maps from the Jiggle Observations

5.13.1. Module Description

This module provides the map-making routines used by the SPIRE photometer pipeline. Jiggle mode observations are processed after denodding and averaging and the resulting map is obtained using the task `NaiveApppMapperTask`. All output maps are North-oriented.

5.13.2. Input Observational Data Products

APPP — **Averaged Pointed Photometer Product** The `NaiveApppMapperTask` takes one Averaged Pointed Photometer Product (APPP) as input. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.13.3. Output Data Products

All map-making tasks returns the map associated to the bolometer array specified by the user. The map is stored as a `SimpleImage`. For a definition, see:

ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/api/herschel/ia/dataset/image/SimpleImage.html

5.13.4. Input Calibration Products Required

there are no calibration files required for the Jiggle Map `NaiveApppMapperTask`

5.13.5. How to use the *Map Making* Module

The module contains 3 mappers: 2 for the scan mode (`NaiveScanMapperTask` and `MadScanMapperTask`) and 1 for the jiggle mode (`NaiveApppMapperTask`).

The input parameters are a context of Photometer Detector Timelines (`PointedProduct`) for the scan mode and an Averaged PointedPhotometer Product for the jiggle mode.

The user should set the `array` keyword to "PSW", "PMW" or "PLW" to select the bolometer array to be processed. The optional keyword `maxmemory` can be used to increase the memory to be allocated for map-making and avoid slowing down by limiting I/O.

The mappers only compute one map at a time, for the bolometer array specifed by the user. The output map is stored as a `SimpleImage` product, for which convenient visualisation tools are available (see examples below) within HCSS.

The mappers use temporary files to store the image coordinates (X,Y). `IOException` will be thrown is these temporary files can not be written or read. Out of memory exception can occur if the size of the map is too large (because of limited resource or incorrect astrometry)

5.13.6. Parameter options

Users can set the following parameters of the Map Making module:

array	**Name of the bolometer array to be processed** The value of this parameter can be "PSW", "PMW" or "PLW". If not set, the default bolometer array is assumed to be "PSW".
resolution	**Pixel size of the output map** The value of this parameter is in arcseconds. If not set, the default values are 6", 10" and 14 for the PSW, PMW and PLW bolometer array.
maxmemory	**Maximum memory in bytes to be used to hold the pointing matrix and the timeline in memory.** If the allocated memory isn't large enough, processing time may be I/O limited. By default, 1GB is assumed.

5.13.7. Examples

Given an Averaged Photometer Pointing Product, the jiggle map is simply obtained in the following way.

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import NaiveApppMapperTask
IA>> mapper = NaiveApppMapperTask()
IA>> psw = mapper(appp, array='PSW')
IA>> Display(psw)
```

5.13.8. Error messages

5.14. Calculating the Flux and Position of a Source

5.14.1. Module Description

The *Point Source Extraction* module gives the position and flux of a point source observed with a 7-point jiggle pattern. As an intermediate result, it gives the outcome of trying to fit a PSF to the signal of every detector.

5.14.2. Input Data Products

The module takes one Pointed Photometer Product (PPP) as input. For the product definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.14.3. Output Data Products

The final output of this module is a Jiggled Photometer Product. For the definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

An intermediate output, called Jiggle Point Source Fit Product (JPSFP), will be produced by the first task of the module. For the definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.14.4. Input Calibration Products Required

No calibration products required.

5.14.5. How to use the *Point Source Flux Density and Position* Module

There are two tasks making up this module, `PointSourceFit` and `SourceFlux`. The former takes a Pointed Photometer Product as input and tries to fit a PSF (currently just a two-dimensional symmetric Gaussian) to the signal of every detector. It outputs a Jiggle Point Source Fit Product. The input product is the only parameter.

The latter task takes the Jiggle Point Source Fit Product as input and produces a Jiggled Photometer Product with the position and flux of the point source, as measured in each of the three detector arrays. There are three optional input parameters, called `positivePixels`, `negativePixels1` and `negativePixels2`. They are `String` arrays taking the name of the pixels where the positive and the two negative images of the source are expected, for each array.

5.14.6. Parameter options

Users can set the following parameters of the Point Source module:

5.14.7. Examples

If `myPpp` is a Pointed Photometer Product with the data you want to analyse, the following example show the steps needed to obtain a Jiggled Photometer Product named `myJpp`:

```
myJpsfp = PointSourceFit()(input = myPpp)
myJpp = SourceFlux()(input = myJpsfp)
```

5.14.8. Error messages

Chapter 6. Processing Data with the Spectrometer Pipeline

6.1. Introduction

A detailed description of the Spectrometer pipeline design can be found in

Fulton T.R., Naylor, D.A., Baluteau, J., Griffin, M., Davis-Imhof, P., Swinyard, B.M., Lim, T., Surace, C., Clements, D., Panuzzo, P., Gastaud, R., Polehampton, E.T., Guest, S., Lu, N., Schwartz, A., Xu, K., The data processing pipeline for the Herschel SPIRE; Imaging Fourier Transform Spectrometer. Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

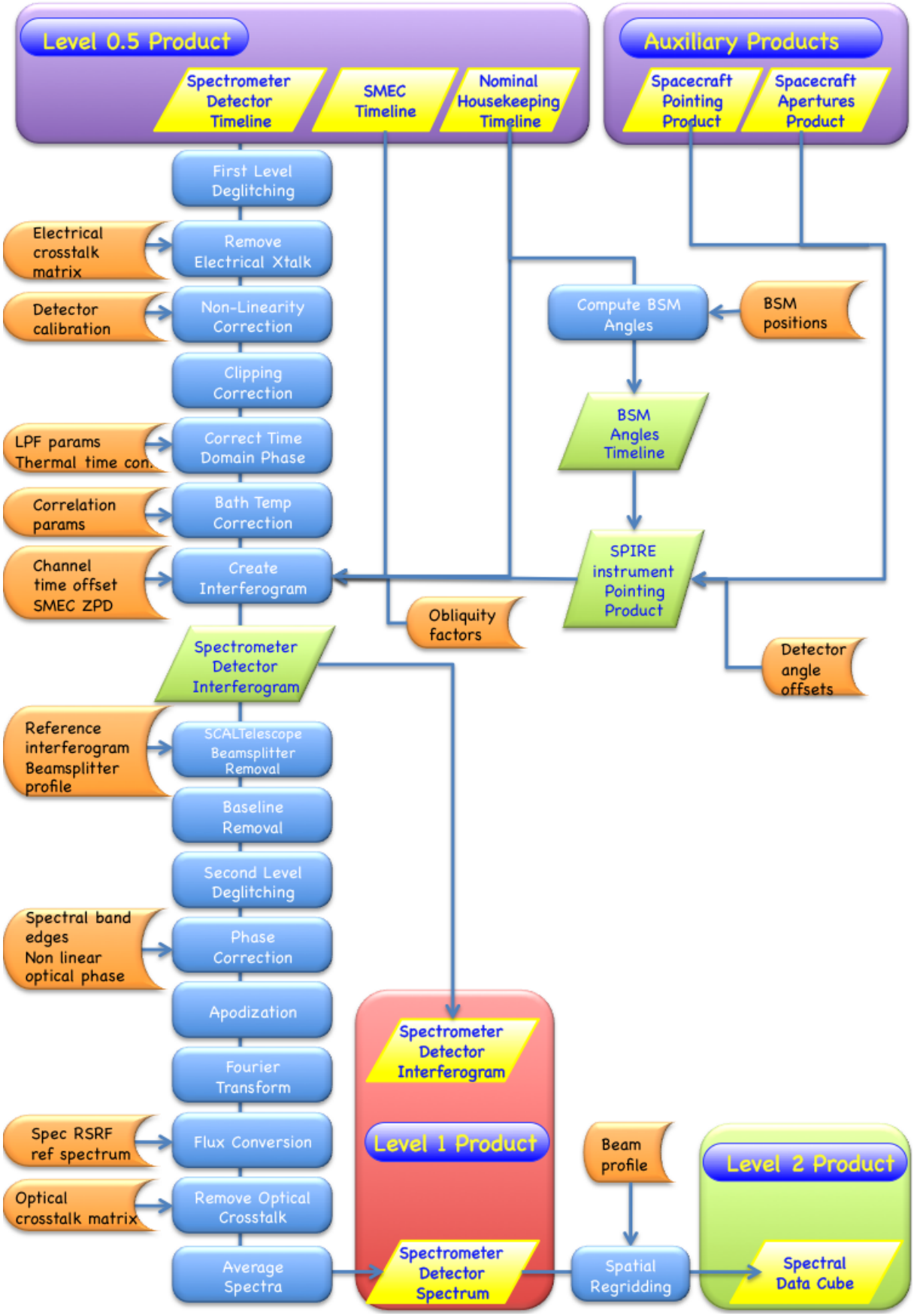

Figure 6.1. Flowchart for the Spectrometer Pipeline.

6.2. Conversion of the BSM telemetry into Angles on the Sky

6.2.1. Module Description

The *Compute BSM Angles* Module converts the sensor signals of the Beam Steering Mirror to physical units (decimal degree in the sky). The method is simple: interpolation from a calibration table containing the two converted angles (Y,Z) versus the two raw angles (chopper angle, jiggle angle). For the Spectrometer, although the BSM is used for the creation of jiggle maps the instrument does not produce a BSM timeline, instead the input comes from the Nominal Housekeeping Timeline (NHKT) which contains the BSM sensor values. This process extracts focal plane Y,Z angles corresponding to the sample time in the NHKT by comparing the sensor signals in the NHKT and BSM Positions Table which contains the Y,Z angles for a given chop and jiggle BSM sensor value. Note that the time in the NHKT is sampled at a lower rate of 1Hz and therefore the corresponding BSM Angles Timeline (BAT) will also be sampled at the same rate. Note that a change request is being implemented to produce a BSM timeline also for the spectrometer.

6.2.2. Input Data Products

NHKT	**Nominal Housekeeping Timeline** The input product contains the time samples and corresponding BSM sensor signal values in the NHKT in raw format in the `chopsenssig` and `jiggsenssig` parameters for each observation building block.

6.2.3. Output Data Products

BAT	**BSM Angles Timeline** The output BAT product contains timelines for the angular offset (in spacecraft Y,Z angles) of the BSM from its rest position on the sky in arc seconds for each observation building block.

6.2.4. Input Calibration Products

BSM Position Table	**BSM Position Table** This Table contains the BSM sensor signals in raw units (in chop and jiggle directions) and angular distance on the sky from its zero position in arcsecs (in spacecraft Y, Z coordinates). There is one table for the photometer and another for the spectrometer, because the reference BSM position can be different.

6.2.5. How to use the *Compute BSM Angles* Module

The module takes a `EdpProduct` object as input. This contains the timeline of the two angles of the mirror (chop and jiggle) in raw data contained in the Nominal House Keeping Timeline (NHKT).

The module also needs a calibration product `BsmPos`. It contains information for converting the raw angles.

The task output is a `BSM Angles Timeline` which contains the timeline of the angles Y and Z (BAT).

6.2.6. Parameter Options

The Compute BSM Angles module does not use any parameters that can be controlled by the user during run-time. The interpolation method is not a parameter.

6.2.7. Examples

Assuming that `nhkt` is an NHKT product and `obs` is an observation context:

```
  bat=calcBsmAngles(nhkt,bsmPos=obs.calibration.spec.bsmPos)
  yang=bat.yangle
  zang=bat.zangle
  t0=bat.sampleTime[0]
  time=bat.sampleTime-t0

  p1=PlotXY(yang,zang,xtitle="Chopper angle (arcsec)",ytitle="Jiggle angle
(arcsec)", titleText="BSM positions ",line=0,symbol=3)
  p1[0].name="BAT"

  p2=PlotXY(time,yang,xtitle="Time (sec)",ytitle="Chopper/Jiggle angle (arcsec)",
titleText="BSM Angle Timeline")
  p2[0].name="Chopper"
  p2[1]=LayerXY(time,zang,name="Jiggle")
  p2.legend.visible=1
```

The process is represented pictorially in the figure below.

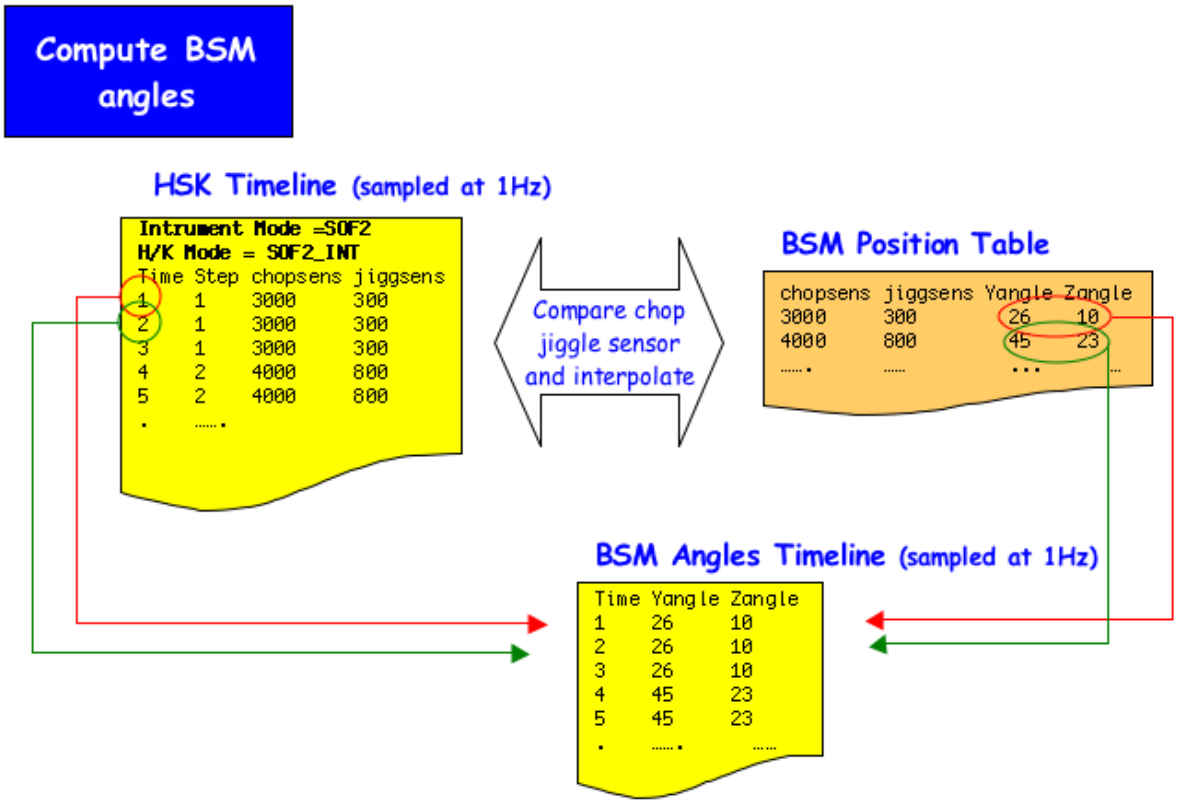

Figure 6.2. Creation of BSM Angles Timeline by the Compute BSM Angles module in the Spectrometer Pipeline.

6.2.8. Error messages

Error in calcBsmAngles Task: the input product is not a beam switch mirror timeline nor a housekeeping. The input product type was checked and found not to be a BSM timeline or a NHKT. The module can only operate on these two product types.

6.3. Deglitching the Detector Timelines

6.3.1. Module Description

The *First Level Deglitcing* Module removes glitches due to cosmic ray hits or other impulse-like events. The basic assumption is that the glitch signature is similar to a Dirac delta function. This process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs a signal free of such glitch signatures.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

6.3.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.3.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains deglitched timelines for each spectrometer detector for each observation building block.

6.3.4. Input Calibration Data Products

No calibration data are needed.

6.3.5. How to use the *First Level Deglitching* Module

6.3.6. Parameter Options

The first level deglitching task employs a complex algorithm, based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent complex processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the extended documentation for this task.

scaleMin, scaleMax, and voices — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMaxe. For example, a feasible range for the practical purposes of the SPIRE FTS is defined by the values 1 and 8 for scaleMin and scaleMax respec-

tively. voices defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the number of voices per range unit is directly proportional to execution time. Both should therefore be kept at the minimum required to accurately determine the Holder index.

thresholdHolder and hMin

thresholdHolder/hMin are the maximum/minimum values of a real data type of the range of slopes which are interpreted as glitch indicators. The range should include -1 and hMin should be smaller than thresholdHolder. For example, a feasible range for the practical purposes of the SPIRE FTS is defined by the values -0.6 and -1.3 for thresholdHolder and hMin respectively. The range defined by thresholdHolder and hMin allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

thresholdCorr

This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This real number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. A feasible value for the practical purposes of the SPIRE FTS is 0.985. The value selected for thresholdCorr allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples flagged as glitches, which are arguably due to the nominal operation of the instrument.

6.3.7. Examples

```
from herschel.spire.ia.pipeline.common.deglitch import DeglitchingTask
deglitch=DeglitchingTask()
sdt=deglitch(sdt,
               scaleMin=1,
               scaleMax=8,
               voices=5,
               hMin=-2.0,
               thresholdHolder=-0.0,
               thresholdCorr=0.985)
```

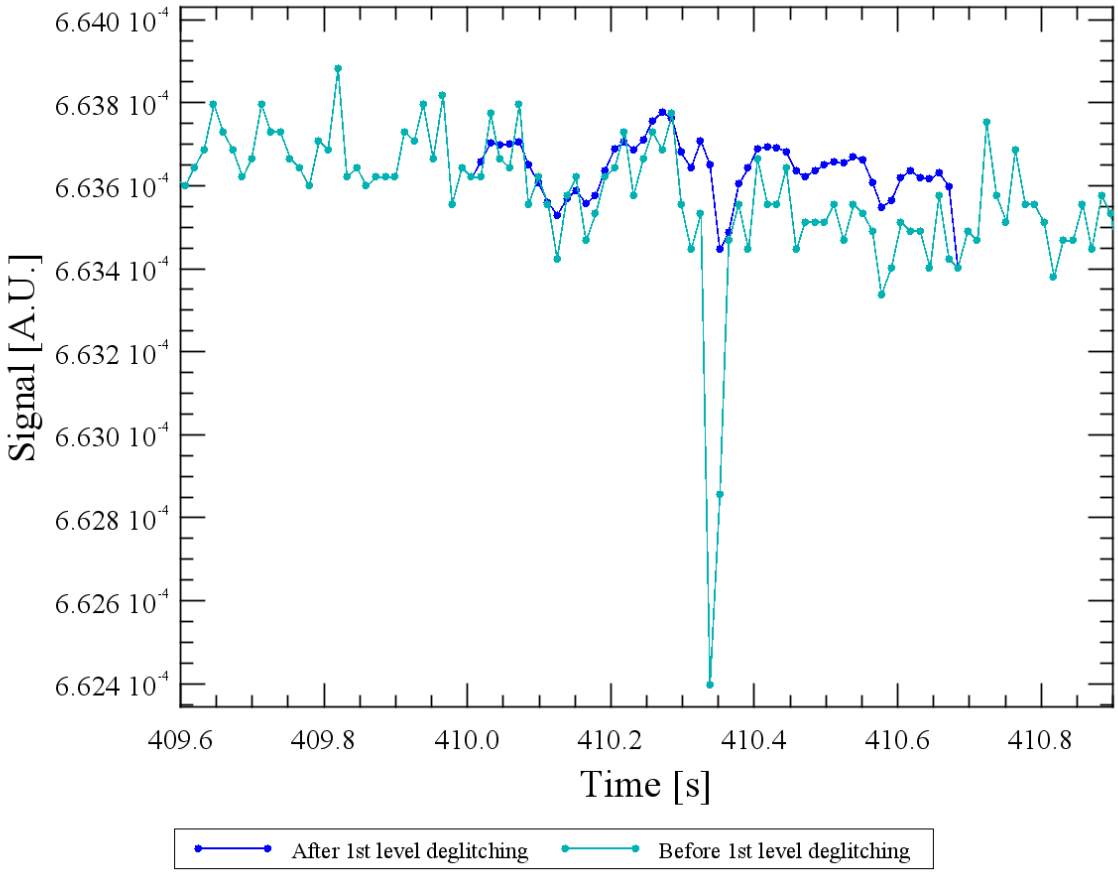

Figure 6.3. An example for glitch removal in a the detector timeline from the SPIRE spectrometer array. An original detector timeline (green) from SLWC1 is shown after removal of the glitch (blue).

6.3.8. Error messages

severe :unable to store output product. The task failed to write the output product.

severe :unable to perform wavelet decomposition. The task failed to identify glitches because the signal could not be decomposed into its wavelet components.

warning : There no signal in the input product. The product contains no valid signal data.

warning :Unable to perform reconstruction. The task failed to reconstruct signal samples.

6.4. Correcting the Detector Timelines for Electrical Crosstalk

6.4.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Spectrometer Detector Timelines (SDT). The task multiplies a crosstalk removal matrix with the signal vector for each instant in time and returns an SDT which are corrected for electrical crosstalk.

6.4.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.4.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block with the signal corrected for electrical crosstalk.

6.4.4. Input Calibration Products

SCalElecCross — **Electrical Crosstalk Matrix** The two electrical crosstalk matrices for the SPIRE spectrometer contain entries for the electrical crosstalk between the detectors of each one of the two detector arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

6.4.5. How to use the *Electrical Cross Talk Correction* Module

It should be noted that, prior to launch, no electrical crosstalk was observed for the spectrometer arrays. Close examination of the in-flight performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this processing task is a placeholder.

6.4.6. Parameter Options

The Electrical Crosstalk Removal module does not use any parameters that can be controlled by the user during run-time.

6.4.7. Examples

Assuming that `sdt` is an SDT product and `obs` an observation context:

```
from herschel.spire.ia.pipeline.common.eleccross import ElecCrossTask

elecCross = ElecCrossTask()

sdt=elecCross(sdt,table=obs.calibration.spec.elecCross)
```

6.4.8. Error messages

Error in ElectricalCrosstalk Correction Task: the input product is not a Photometer or Spectrometer Detector timeline The electrical crosstalk removal task requires a detector timeline product as input (either for the photometer or the spectrometer). The module cannot operate on a product of any other type.

Error in ElectricalCrosstalk Correction Task: the input product is already corrected. The electrical crosstalk removal task should only be applied once during data processing. An error is thrown if the module is applied more than once and the module is not executed.

6.5. Correcting for Clipped Data

6.5.1. Module Description

The 16-bit Analogue-to-Digital Converter (ADC) to read the detector voltages into SPIRE's on-board digital processing unit imposes a minimum (0) and maximum (65535) value to the detector readings. The samples at the extremes are flagged as truncated by the Check ADC Flags and Truncation module at an earlier stage of the pipeline. The Clipping Correction task reconstructs those samples of the spectrometer detector timeline (SDT) which are flagged as truncated. Samples are reconstructed by using a polynomial fitting for each data range affected by truncation.

6.5.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.5.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block.

6.5.4. Input Calibration Products

No calibration data are needed.

6.5.5. How to use the *Clipping Correction* Module

6.5.6. Parameter Options

The Clipping Correction task employs a simple algorithm, based on fitting data with a high order polynomial function. Users are strongly discouraged from changing the default value of 5 for the highest order of the polynomial function.

number — The parameter "number" defines the number of points used for the polynomial fitting to either side of a truncated data range. Overall, 2 x "number" points will be used for the polynomial fitting. "number" should be an integer greater than 1.

6.5.7. Examples

```
from herschel.spire.ia.pipeline.spec.clip import ClippingTask

clipCorrection=ClippingTask()

sdt=clipCorrection(input=sdt,number=5)
```

6.5.8. Error messages

severe : unable to store output product The task failed to write the output product.

warning : the optimized parameter number of fitting points is not used : "xx" that may affect the quality of the reconstruction. The risk that the algorithm introduces spectral artifacts is higher if the user selects the number of points used for the polynomial fitting to be different from its default value.

warning : clipping not be done for the (xx) point of the sample (first point of the clipped area) clipped area excess 9 consecutive point or clipped area at the begining or the end of the signal. Clipped samples cannot be corrected if they are at the very beginning or end of the timeline since a minimum number of data points are required for the high order polynomial fitting. Samples cannot be corrected either if more than 9 consecutive samples are clipped.

warning : reconstruction not be done for the "xx" point of the sample (first point of the clipped area of "yy" consecutive points) polynomial fitting fails. Data samples were not reconstructed because the polynomial fitting routine failed for the specificed truncated range.

warning : some clipped samples are glitched "xx" and they are not be corrected. Data samples which have been identified as glitches are not reconstructed even if they are clipped. The polynomial fitting would not be suitable in this case and lead to invalid results.

6.6. Correcting for Time Domain Phase Shifts in the Data

6.6.1. Module Description

The *Time Domain Phase Correction* task corrects for the filtering effects due to the read-out electronics and the thermal behaviour of the bolometer detectors.

The SPIRE spectrometer detector chain contains a 6-pole Bessel low pass filter (LPF) and an additional RC LPF. In addition to the electronic LPF, the thermal behaviour of the SPIRE bolometers is modeled as an RC LPF with a detector-specific time constant, τ. The LPFs will affect the magnitude of the signal recorded by the SPIRE detectors and induce a phase shift to the recorded signal. The phase shift was characterized during ground-based test campaigns with one free parameter, the detector-specific time constant τ. The time constants are retrieved from a calibration product and a time domain phase correction function is derived accordingly. The measured detector timelines are corrected by a convolution with the derived time domain phase correction function. The edges of the timelines, invalidated by the convolution operation are truncated.

6.6.2. Input Observational Data Products

SDT **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.6.3. Output Data Products

SDT **Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block, corrected for the low pass filtering.

6.6.4. Input Calibration Data Products

SCalSpecLpfPar — **Spectrometer Low Pass Filter Parameters** This calibration product contains the parameters of the low pass filters of the read-out electronics of the SPIRE spectrometer.

SCalSpecDetTimeConst — **Spectrometer Detector Time Constants** This calibration product contains the thermal relaxation time constant τ per detector. The time constants range between 3.3 and 13.4 ms.

6.6.5. How to use the *Time Domain Phase Correction* Module

6.6.6. Control parameters

The Time Domain Phase Correction module does not use any parameters that can be controlled by the user during run-time.

6.6.7. Examples

```
  from herschel.spire.ia.pipeline.spec.phase import TimeDomainPhaseCorrection
  timeDomainPhaseCorrection=TimeDomainPhaseCorrection()
  sdt=timeDomainPhaseCorrection(sdt=sdt,
                                  lpfPar = obs.calibration.spec.lpfPar,
                                  chanTimeConst =
obs.calibration.spec.chanTimeConst)
```

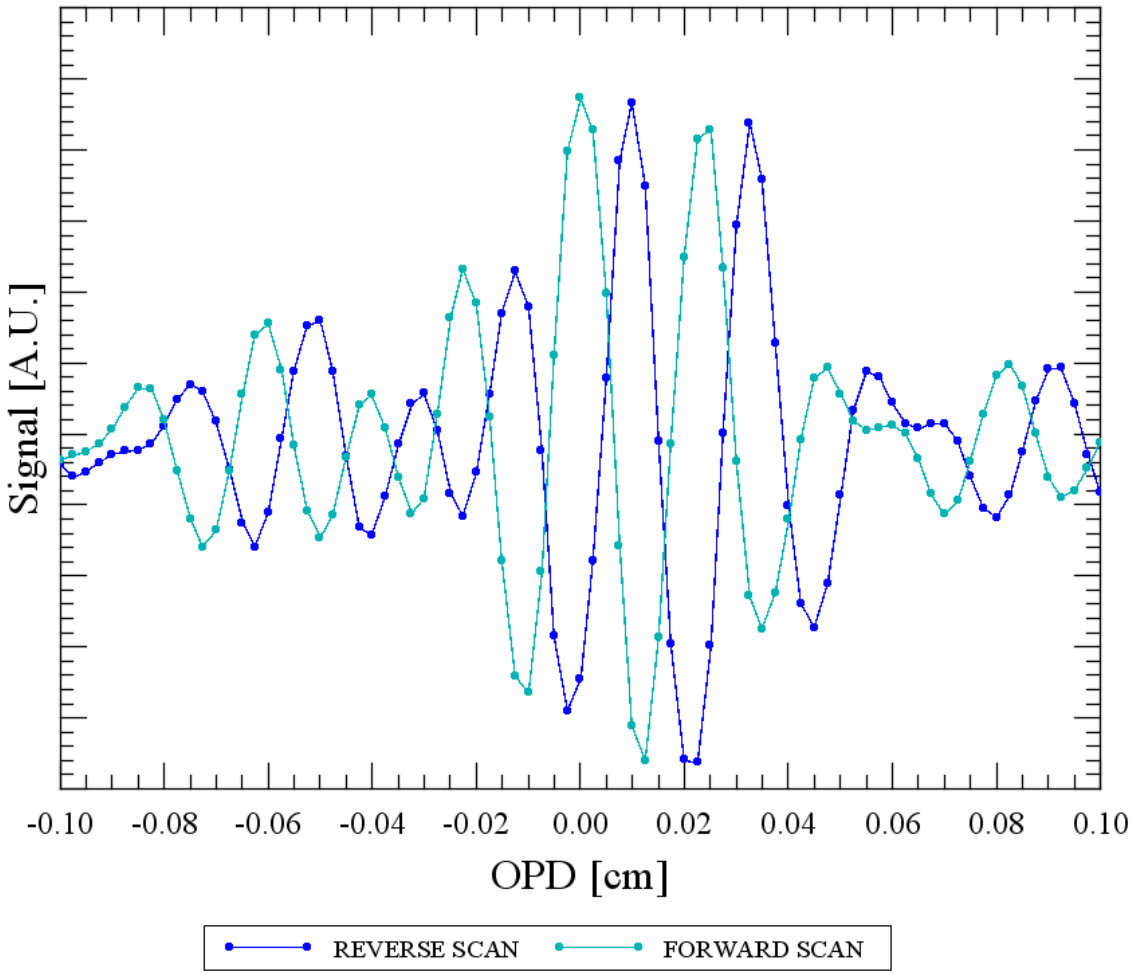

Figure 6.4. Forward and reverse interferograms do not line up well without applying time domain phase correction. A forward (green) and a reverse (blue) scan are slightly shifted with respect to one another.

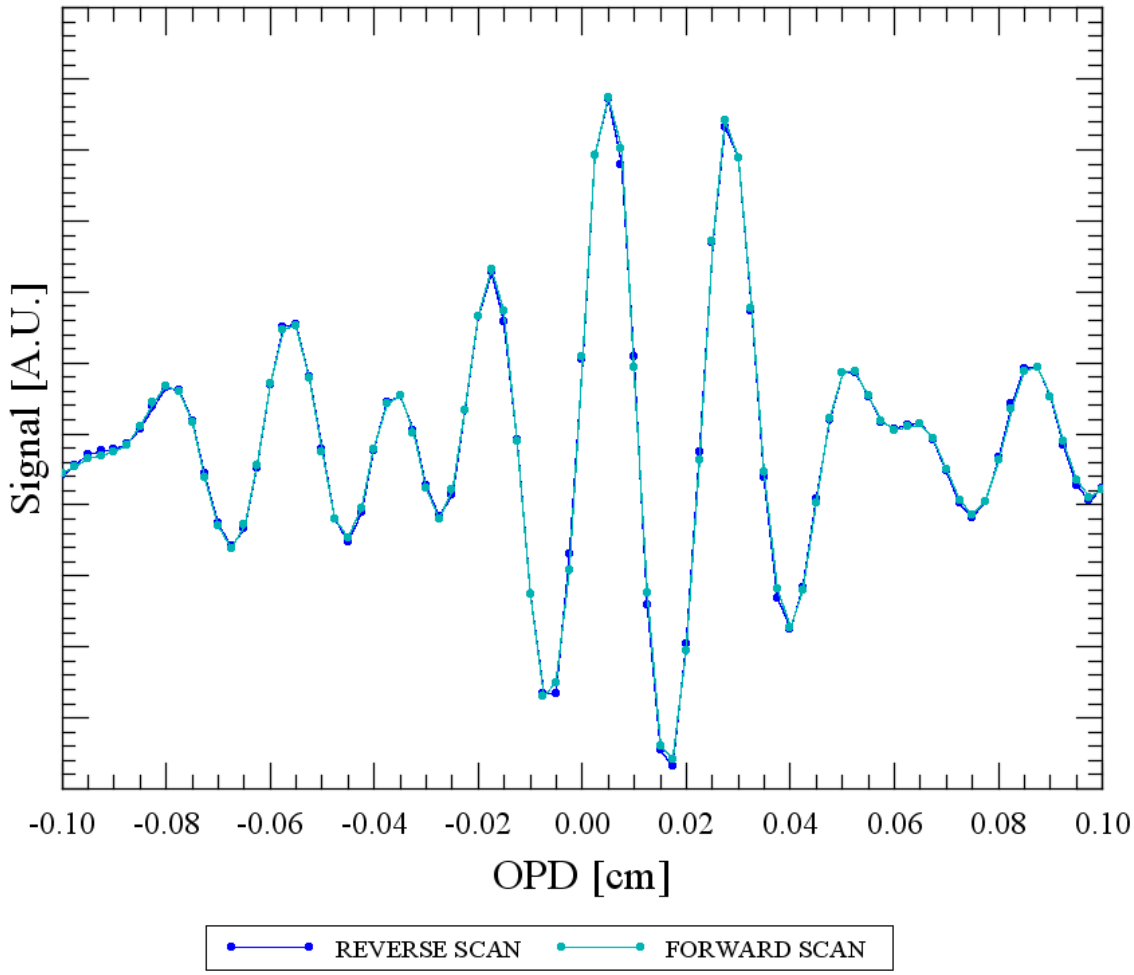

Figure 6.5. Forward and reverse interferograms line up much better when applying time domain phase correction to the Spectrometer Detector Timelines. Forward (green) and reverse (blue) scan.

6.6.8. Error messages

There are no specific error messages defined for this task.

6.7. Applying the Non-Linearity Correction to the Data

6.7.1. Module Description

The *Non-Linearity Correction* module applies a correction for the nonlinear bolometer responsivity to a spectrometer detector voltage timeline V (SDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector channel in a calibration product (SCalSpecNonLinCorr). The result is a voltage timeline SDT where the voltage is directly proportional to the power incident on a detector.

6.7.2. Input Data Products

SDT	**Spectrometer Detector Timeline** The input SDT with its bolometer signal converted to electrical units (e.g. V).

6.7.3. Output Data Products

SDT	**Spectrometer Detector Timeline** The output SDT product with bolometer signal in units of Volts corrected for any non-linearity.

6.7.4. Input Calibration Products

ScalSpecNonLinCorr	**Spectrometer Non-Linearity Correction** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the limit bolometer voltages for which the calibration is valid.

6.7.5. How to use the *Non-Linearity Correction* Module

6.7.6. Parameter Options

The Nonlinearity Correction module does not use any parameters that can be controlled by the user at run-time.

6.7.7. Examples

Assuming a detector timeline of SPIRE spectrometer data processed to Level 0.5, with bolometer signal data in units of volts `sdt` and `obs` is an observation context:

```
 from herschel.spire.ia.pipeline.spec.nonlin import SpecNonLinearityCorrectionTask

 nonlinearCorrection = SpecNonLinearityCorrectionTask()

 linearizedProduct = nonlinearCorrection(sdt,
table=obs.calibration.spec.nonLinCorr)
```

6.7.8. Error messages

Error in NonLinearCorrection Task: the input product is not a Photometer or Spectrometer Detector timeline. This module can only process a detector timeline product, either from the photometer or the spectrometer.

Error in Flux Conversion Task: the input product is already corrected. This module should only be applied once during data processing. An error is thrown if the module is applied more than once.

severe: SignatureException in setting the result!! The module issues a severe warning if it was not possible to set the calculated values.

Error messages generated: If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

6.8. Removing Correlated Noise due to bath temperature fluctuations from the Data

6.8.1. Module Description

The *Bath Temperature Fluctuation Correction* module subtracts low frequency (< 1Hz) noise, caused by variations of the detector array bath temperature. The module is based on the empirical correlations between detectors and thermistors (or, in case of high bias voltages when thermistors are saturated, the correlations between detectors and dark pixels). The module shall be run after the nonlinearity correction module.

6.8.2. Input Data Products

SDT	**Spectrometer Detector Timeline** The SDT with the bolometer signal converted to electrical units (e.g. V) directly proportional to the incident power.

6.8.3. Output Data Products

SDT	**Spectrometer Detector Timeline** SDT in the units of V, corrected for temperature drift.

6.8.4. Input Calibration Products

ScalSpecTempDriftCorr	**Temperature Drift Correction Parameter Table** A product containing for each detector array, a thermistor selection flag, the base voltage values for each thermistor, the errors for those values, and a validity calibration flag for those values. It contains for each bolometer channel, correction coefficients A1, A1 error, B1, B1 error, AB1 flag (all for thermistor 1) and similar values for thermistor 2.

6.8.5. How to use the *Bath Temperature Fluctuation Correction* Module

6.8.6. Parameter Options

The bath temperature correction module does not use any parameters that can be controlled by the user during run-time.

timeSpan	This is the smoothing timespan in seconds. Its default value is 5.

6.8.7. Examples

Assuming a detector timeline of SPIRE spectrometer data, `sdt` with bolometer signal data in units of V and processed by the spectrometer Non-Linearity Correction Module and an observation context `obs`:

```
from herschel.spire.ia.pipeline.common.tempdrift import
TemperatureDriftCorrectionTask

tempDrift = TemperatureDriftCorrectionTask()
```

```
sdt=tempDrift(data=sdt, table=obs.calibration.spec.tempDriftCorr, 5)
```

6.8.8. Error messages

Error in TemperatureDriftCorrection Task: the input product is not a Photometer or Spectrometer Detector timeline. This module can only process a detector timeline product, either from the photometer or the spectrometer.

Error in TemperatureDriftCorrection Task: the input product is already corrected. This module should only be applied once during data processing. An error is thrown if the module is applied more than once.

Error in TemperatureDriftCorrection Task: biasMode parameter is incorrectly specified in calibration product.

??

Error in TemperatureDriftCorrection Task: biasMode parameter is not specified in calibration product.

??

Error in TemperatureDriftCorrection Task: "xx" parameter is not specified or is incorrectly specified in for array "SSW/SLW" of the calibration product.

??

Error in TemperatureDriftCorrection Task: timeSpan parameter is not specified in calibration product.

??

Error in TemperatureDriftCorrection Task: VT02 parameter is not specified in calibration product. .

??

Error in TemperatureDriftCorrection Task: VT02 parameter is not specified in calibration product..

??

Error in TemperatureDriftCorrection Task: VT01Flag parameter is not specified in calibration product.

??

Error in TemperatureDriftCorrection Task: VT02Flag parameter is not specified in calibration product.

??

Error in TemperatureDriftCorrection Task: timeSpan parameter is not specified in calibration product.

??

Error in TemperatureDriftCorrection Task: biasSwitch failure - program logic error.

??

6.9. Creating the Interferograms from the Timeline Data

6.9.1. Module Description

A single building block of a SPIRE spectrometer observation in scanning mode consists of a series of scans of the spectrometer mechanism (SMEC) while the instrument is pointed at a given target. The sampling of the SPIRE spectrometer detectors and the spectrometer mechanism is not synchronized; the two subsystems are sampled at different rates and at different times. In order to derive the source spectrum from the measured data, the spectrometer detector samples must be linked with the position of the SMEC in the form of interferograms, i.e. signal as a function of optical path difference (OPD). Additionally, the spectrometer detector signal timelines are interpolated onto timelines where the SMEC positions are equidistantly spaced to ensure accurate transformation of the interferogram with the Discrete Fourier Transform. First, the SMEC timeline is interpolated from one that is non-equidistant in position to one that is equidistant in position. Then, the detector signal timelines are interpolated onto the times which correspond to the equidistant SMEC position grid. In addition, the mean value of the pointing timeline as derived from the whole observation building block is assigned to the output SDI product.

6.9.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The SDT contains the measured signal sample timelines for each spectrometer detector in units of Volts. The units are expected to be identical for all channels.

SMECT — **Spectrometer Mechanism Timeline** This product contains the timelines for the telemetry parameters related to the spectrometer mechanism (SMEC). The unit of the coarse and fine optical encoder readings is expected to be centimetre.

PT — **Pointing Timeline** This product contains the timelines for the positions in the sky of the line of sight of the SPIRE FTS.

NHKT — **Nominal Housekeeping Timeline** This product contains the timelines for the nominal housekeeping telemetry parameters. The housekeeping parameters used by this module are:

- **SCANSTART** This quantity contains the commanded start position for the SMEC for the observation in centimetres. All entries in this timeline should be identical.
- **SCANEND** This quantity contains the commanded end position for the SMEC for the observation in centimetres. All entries in this timeline should be identical.
- **SCANS** This quantity contains the value of the scan number. This quantity is expected to be unitless and its values should only decrement over the course of the observation.

6.9.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** The SDI is an output product that contains the interferograms for each spectrometer detector for each scan of the observation. The signal quantities of this output product will be in units of Volts - identical to the

signals of the input SDT product. The quantities of the OPD columns of this product will be in units of centimetres.

6.9.4. Input Calibration Products

calSmecZpd	**Zero Path Difference** This calibration product contains the values of the position of zero path difference (ZPD) for each SPIRE spectrometer detector. There are two entries for each detector; one entry gives the SMEC optical encoder value for ZPD in units of centimetres, the other entry gives the LVDT; value for ZPD.
calChannelStepFactorTable	**Optical Encoder to Optical Path Difference** This calibration product contains the conversion factors that relate a step of the SMEC encoder to a step in OPD. Nominally, this conversion factor is equal to 4 for a Mach-Zehnder FTS. For the SPIRE FTS this conversion factor will be different for each detector and depend on the off-axis angle. Each entry is expected to be unitless.
calSpecChanMask	**Spectrometer Detector Channel Mask** This calibration product contains entries for each spectrometer detector to indicate whether the detector is known to be either noisy or dead. "Dead" detectors will be omitted from the output SDI product.
calSpecChanTimeOff	**Spectrometer Channel Time Offset** This calibration product contains the time offsets between the frametime and the readout time for the spectromter detectors in the SDT product. The quantities in this calibration product are expected to be in units of seconds.

6.9.5. How to use the *Interferogram Creation* Module

6.9.6. Control parameters

Users can set the following parameters of the Interferogram Creation module:

interpolType	This parameter specifies the type of interpolation to be performed. The user has two choices: spline or oversampled. The default value is spline, indicating that cubic spline interpolation will be used. This simple interpolation method has the advantage of being robust and has been found not to degrade the quality of the data from the SPIRE FTS. If this input parameter is set to "oversampled" each interpolation will be performed via a linear fit after a fine super-sampling of the data via the Fourier domain.

6.9.7. Examples

Assuming `sdt` to be a detector timeline of SPIRE spectrometer data processed to Level 0.5, with bolometer signal data in units of Volts, `smect` to be a SMEC timeline, `spp` to be a SPIRE pointing product and `obs` to be an observation context:

```
from herschel.spire.ia.pipeline.spec.ifgm import *

createIfgm = CreateIfgmTask()

sdi = createIfgm(sdt=sdt,
```

```
                        smect=smect,
                        hkt=nhkt,
                        spp=spp,
                        interpolType="spline",
                        calSmecZpd=obs.calibration.spec.smecZpd,
calChannelStepFactorTable=obs.calibration.spec.smecStepFactor,
                        calSpecChanTimeOff=obs.calibration.spec.chanTimeOff)
```

6.9.8. Error messages

error: SCANSTART/SCANEND not repeating. This module will throw an exception if the value of the housekeeping parameters SCANSTART/SCANEND change. These parameters should specify the commanded scan start and end positions and should therefore not change during a building block.

severe: Input contained no good scans, SDI is empty. This module will issue a severe message if the resulting SDI product contains no valid scans.

severe: x "min" - x "min - 1" is negative. Note this difference occurs outside the first "xx" points (oversampling factor)".

severe: x "max" - x "max - 1" is negative. Note this difference occurs outside the last "xx" points (oversampling factor).

severe: SDT signals not of type Double or Float. Cannot continue. This modules will issue a severe message if the input SDT product does not contain real numbers in the signal columns, either of data type Double or Float.

severe: No good minimum/maximum encoder positons for observation.

severe: endPos (lesseq) startPos, which should be impossible. This module will issue a severe message if the start position of the interferograms was determined to be larger than their end position.

severe: NHKT.SCANRES not UNIQUE. Cannot continue. This module will issue a severe message if the NHKT indciates a varying resolution of the observation. In this case, the module will not attempt to use the NHKT to derive the resolution of the observation.

warning: Could not derive a commanded res (res == null). This module will issue a warning if it was not possible to determine the commanded resolution.

warning: Poorly formed data. This module will issue a warning if the signal column for any of the detector channels in the input SDT is null or contains NaN; values. In this situation, execution continues as normal and data from the offending channel are not processed further.

warning: Problem creating I(x) for channel: "xx". Skipping ... This module will issue a warning if it was not possible to calculate a valid interferogram for a specific detector.

warning: Units missing: assuming microns for encoderCoarse, nanometers for encoderFine This module will issue a warning if the SMEC timeline does not specify the units of the optical encoder readings. Default units are assumed as stated in the warning message.

warning: Scan "xx" has bad length and has been flagged for removal. This module will issue a warning if a particular scan was found to be of abnormal length, i.e. more than 5% deviation from the nominal start and end positions.

warning: No SpecSmecStepFactor calibration product provided. Using default step factor: 4. This module will issue a warning if the SpecSmecStepFactor calibration product is missing and the default step factor is used instead.

warning: Step Factor not found for "xx". Using default step factor: 4. This module will issue a warning if the SpecSmecStepFactor calibration product does not contain a step factor for the specified detector. The default value of 4 is used instead.

warning: No SpecSmecZpd calibration product provided. Using default ZPD : "xx" cm. This module will issue a warning if the SpecSmecZpd calibration product is missing and the default position for ZPD is used instead.

warning: ZPD not found for "xx". Using default ZPD : "yy" cm. This module will issue a warning if the SpecSmecZpd calibration product does not contain a ZPD value for the specified detector. The indiciated default value is used instead.

warning: start position: "xx" > minimum MPD: "yy". Removing extra points. This module will issue a warning if the start position of the created interferograms is higher than the expected. Interferograms will contain fewer points.

warning: end position: "xx" (lessthaeq) maximum MPD: "yy". Removing extra points. This module will issue a warning if the end position of the created interferograms is lower than the expected. Interferograms will contain fewer points.

warning: Poorly formed SCANFSPEED/SCANRSPEED. warning: SCANFSPEED != SCANRSPEED. This module will issue a warning if the arrays containing the scan speeds from the housekeeping data either indicate varying commanded SMEC speeds throughout the observation or different commanded SMEC speeds for forward and reverse scans.

6.10. Removing the Telescope and Calibration Source Background from the Data

6.10.1. Module Description

The SPIRE FTS measures a signal from the astronomical source but also from the warm Herschel telescope and the Spectrometer Calibration Source (SCal). The *SCAL and Telescope Correction* Module aims to eliminate the contribution from the telescope and SCal by subtracting a deep reference measurement of a dark region of the sky.

6.10.2. Input Data Products

SDI	**Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observation building block.
NHKT;	**Nominal HouseKeeping Product** This product contains temperature timelines for SCal.

6.10.3. Output Data Products

SDI	**Spectrometer Detector Interferogram** The output SDI product contains the corrected interferograms for each spectrometer detector for each scan of the observation building block.

6.10.4. Input Calibration Products

SpecInterRefList	The SpecInterRefList Calibration products contain interferograms from a deep observation of a dark region in the sky for each spectrometer detector and the two directions (forward and reverse) of the linear translation stage (SMEC) of the SPIRE FTS. There are different ediions depending on the bias mode (nominal or high).

6.10.5. How to use the *SCAL and Telescope Correction* Module

6.10.6. Parameter Options

The Telescope and SCal Correction module does not use any parameters that can be controlled by the user at run-time.

6.10.7. Examples

Assuming an `sdi` spectrometer detector interferogram product and `obs` is an observation context:

```
  from herschel.spire.ia.pipeline.spec.scal import *
  telScalCorrection = ScalTask()
  sdi = telScalCorrection(sdi=sdi,

sdical=obs.calibration.spec.interRefList.getProduct(sdi.startDate),
      nhkt=nhkt)
```

6.10.8. Error messages

severe :unable to store output product. The module failed to write the output product.

warning : scals temperature fluctuations too large. The module issues a warning if the temperature of SCal varies too strongly during the observation to ensure an accurate removal of its spectral signature.

warning : calibration product sdical is not correct return = sdi enter. warning : Calibration product sdical is not compatible with Enter product return = sdi enter. The module issues a warning if the input SDI was not corrected but returned as output without changes due to an non-suitable calibration product.

6.11. Removing the Baseline from the interferograms

6.11.1. Module Description

The intensity incident on the SPIRE spectrometer detectors can be separated into two components: a component that is constant as a function of OPD and a component that is modulated as a function

of OPD. As the first baseline term does not contain relevant spectral information, it may be removed without affecting the source spectrum. The second term can also be removed if only low frequency components are used. On a detector-by-detector and scan-by-scan basis, the baseline correction algorithm evaluates and removes the baseline of the interferogram. The task offers two options to characterize the baseline: Either a polynomial fit or the inverse Fourier transform of the spectrum including only low frequencies.

6.11.2. Input Data Products

SDI **Spectrometer Detector Interferogram** The input SDI product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.11.3. Output Data Products

SDI **Spectrometer Detector Interferogram** The output SDI product contains the corrected interferograms for each spectrometer detector for each scan of the observational building block.

6.11.4. Calibration Data Products

No calibration data are needed.

6.11.5. How to use the *Baseline Correction* Module

6.11.6. Parameter Options

Users can set the following parameters of the Baseline Correction module:

type This parameter determines the type of baseline fitting to perform. The user has two options: "polynomial" or "fourier". Set this parameter to "polynomial" to perform a least-square minimized polynomial fit or to "fourier" to fit the baseline with the low frequency Fourier components of the spectrum that corresponds to the interferogram. Both options are of comparable quality in the sense of removing any baseline variation in the interferograms and not significantly affecting the spectral content in the optical passbands. In fact, the method based on the low frequency Fourier components is slightly better in these terms. Spectral artifacts are usually not significant, i.e. small compared to instrumental noise. However, polynomial fitting is the default fitting type because it will prepare the data better than Fourier fitting for the second order deglitching step [TBC].

Note that both algorithms will fail when applied to interferograms which are truncated and have not been repaired by the clipping correction module.

degree If the fitting type is "polynomial" then "degree" specifies the order of the polynomial function to fit to the baseline of the input interferograms. The default value is 4. The degree of the polynomial function should be even because of the even symmetry inherent to the FTS. An order of 2 approximates the baseline slightly worse and introduces stronger spectral artefacts in the optical passbands. Polynomial functions with orders larger than 4 are possible but do not usually improve the reults significantly.

threshold If the fitting type is "fourier" then "threshold" specifies the maximum frequency in inverse centimeters to include when using the Fourier components to fit to the baseline. 4 cm^{-1} has been found to be a good value for the specific purposes of the SPIRE FTS in terms of achieving a good fit to the baseline and introducing spectral artefacts below the noise.

If interferograms display unusual artefacts such as temporary deviations from the baseline (sudden steps, pepper and salt noise) then a Fourier type fitting with a high threshold (e.g. 10 cm^{-1}) may be effective at removing such unwanted features.

6.11.7. Examples

```
from herschel.spire.ia.pipeline.spec.baseline import BaselineCorrection

baselineCorrection = BaselineCorrectionTask()

sdi=baselineCorrection(sdi=sdi,

                       type="polynomial",

                       degree=4)
```

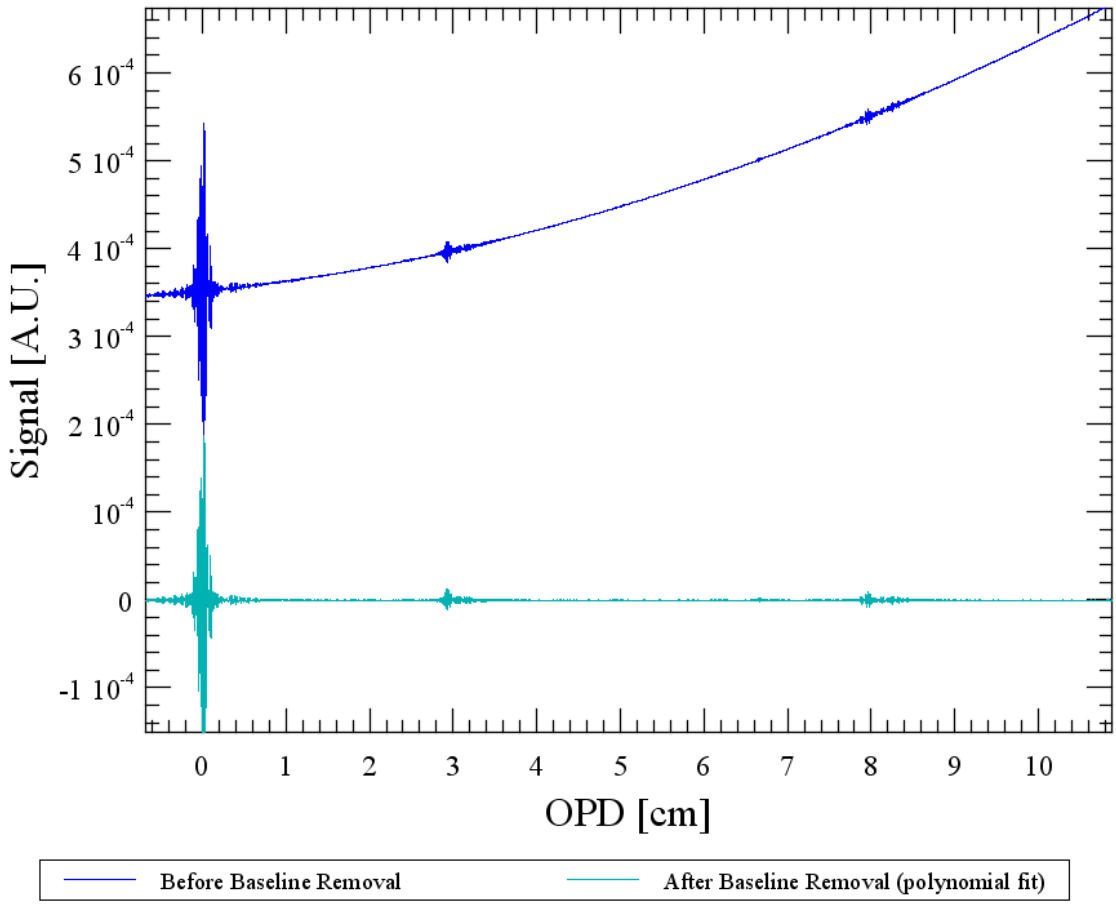

Figure 6.6. An example for removing the baseline with a 4th order polynomial fit to the interferogram. The interferogram before (blue) and after (green) baseline removal.

6.11.8. Error messages

error: Invalid baseline correction type specified: "xx" The type control parameter must be either omitted and set to one of the following two strings: "polynomial" (default) or "fourier".

error: OPD units are "xx". Expected units of cm. The x-axis of the interferograms is OPD. If no units are specified for the OPD, the assumption is made that the values are in centimeters. If units are indeed specified for the OPD, then it must be in units of centimeters. Otherwise the module will throw an exception.

6.12. Removing Glitches from the Interferograms

6.12.1. Module Description

The *Second-Level Deglitching* Module identifies and removes glitches in the interferograms for each spectrometer detector. On an OPD-position-by-OPD-position basis, the samples from one scan are compared to those from all other scans in the same observation. Two different methods are available to compare the data and flag outliers. The baseline approach is to use the median of the spectral data across scans and a threshold factor times the standard deviation or the median absolute deviation to define a range for outliers. The alternative approach uses a window of a user-defined length within which the standard deviation is computed and outliers are flagged based on a standard deviation or median absolute deviation. The scan with the largest absolute deviation will then be discarded and the procedure repeated until no further outliers are flagged. The flagged samples are then replaced by the average of the non-glitch samples from the other observed interferograms at that position.

6.12.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** The input SDI product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.12.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** The output SDI product contains interferograms for each spectrometer detector for each scan of the observational building block which have been corrected for glitches.

6.12.4. Input Calibration Products

N/A

6.12.5. How to use the *Second-Level Deglitching* Module

6.12.6. Parameter Options

deglitchType — This optional parameter specifies the method used to identify glitches. Values may be either "STD", "MAD", "STD_WINDOW", or "MAD_WINDOW". If "STD" or "MAD" are selected then glitches are identified by a 1st order outlier criterion based on the difference of a given sample from the median value across all scans at a given Optical Path Difference (OPD) in units of either standard deviation or median absolute deviation. If the deviation is greater than a user-defined threshold (see keyword below) then the sample is flagged as a glitch. This commonly used approach

to outlier detection is suitable for large iteration numbers. If "STD_WINDOW" or "MAD_WINDOW" are selected then glitches are identified by a 2nd order outlier criterion based on the standard deviation interferogram across all scans. If, within the user-defined window (see keyword below), the standard deviation interferogram contains outliers in the sense of a threshold factor (see keyword below) times either standard deviation or median absolute deviation then the OPD location is identified as containing a glitch. The scan with the largest deviation from the median is then flagged as a glitch and the procedure is iterated until no more glitches are found or only two scans are left. All of the deglitching algorithms require at least three valid scans. The algorithms will ignore samples flagged as unusable. By default, the module will decide itself which method to use in order to identify glitches. This decision follows the above guideline [TBC].

thresholdFactor — This parameter specifies the factor by which the standard deviation or the median absolute deviation is multiplied to define the range of the corridor outside of which samples are flagged as outliers. This parameter applies to all four types of deglitching algorithms, if in slightly different ways for 1st and 2nd order outlier detection. A larger threshold will result in less sensitive glitch detection and fewer data points which are incorrectly flagged as glitches (false positives or ghosts). A typical threshold factor for large sample sizes is a value of 3. Note that the threshold factor for algorithms using the median absolute deviation, the threshold factor must be multiplied by 1.4826 in order to make the results comparable with algorithms using the standard deviation. The module sets default threshold values based on the number of scans in the observation if the user does not specify the threshold value. The default value is 3.2905 for "STD" and "MAD" [TO BE CHANGED], setting the expectation for false positives in white noise to 0.1%. The default values for the other two deglitching schemes depend on the number of scans in a given direction. The default values for "STD_WINDOW" are 4.0, 3.5, 3.0, and 2.5 for 2, 3, 4 - 8, and >9 scans in one direction. The default values for "MAD_WINDOW" are 4.5, 4.0, 3.5, and 3.0 for 2, 3 - 4, 5 - 8, and >9 scans in one direction. Note that glitch identification is currently not set up to deal with the case of only two scans in a given direction.

windowSize — This parameter of type integer specifies the window size for the deglitching types "STD_WINDOW" and "MAD_WINDOW". It must be an odd number in order to define a symmetrical data range around the sample to be analyzed. Its value must be equal to three or larger. The default values for the deglitching types "STD_WINDOW" and "MAD_WINDOW" are 41 and 33 respectively. It is recommended to keep this parameter smaller than the width of the center burst of the measured interferograms. This parameter has no bearing on the other two types "STD" and "MAD".

identifyGlitches — This boolean parameter with default value True specifies whether the task attempts to identify glitches. This option was made available to allow for purely manual glitch detection.

correctGlitches — This boolean parameter with default value True specifies whether the task attempts to replace flagged samples.

6.12.7. Examples

```
IA>> from herschel.spire.ia.pipeline.spec.ifgm.deglitch import DeglitchIfgm

IA>>

IA>> deglitchIfgm=DeglitchIfgm()

IA>> sdi=deglitchIfgm(sdi=sdi,
```

```
                          deglitchType="MAD_WINDOW",

                          windowSize = 33,

      thresholdFactor=3.0)

IA>>
```

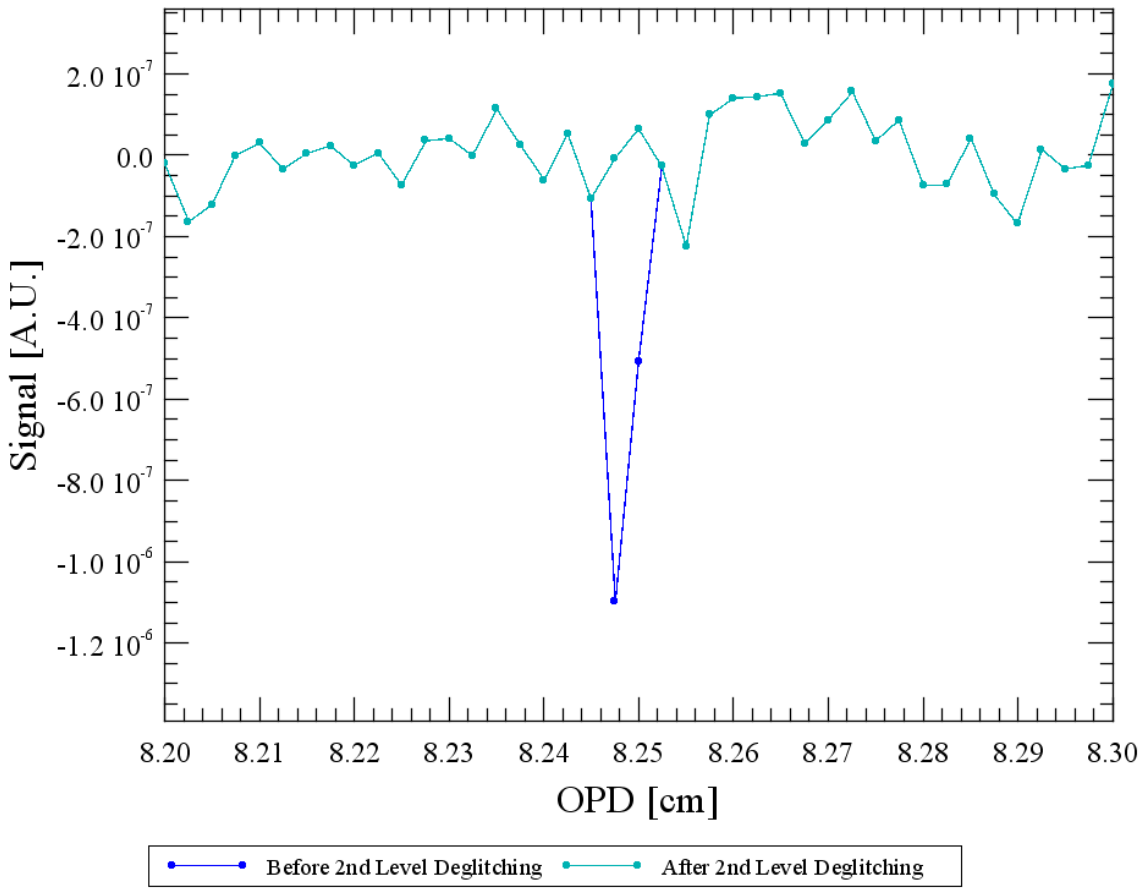

Figure 6.7. An example for glitch removal in an interferogram. An interferogram before (blue) and after glitch removal (green).

6.12.8. Error messages

Invalid threshold parameter specified. A non-positive threshold is mathematically non-sensical. An exception is thrown if this input parameter is set to a non-positive value.

Not enough scans. If not enough forward or reverse scans are available to perform the selected glitch identification algorithm then there be a warning to the user on the Windows console.

6.13. Phase Correction

6.13.1. Module Description

The *Phase Correction* module corrects for any asymmetries in the interferogram about the position of Zero Path Difference (ZPD). The symmetry of the optical layout of a Fourier transform spectrometer (FTS) theoretically implies that the recorded interferograms exhibit even symmetry. The Fourier

transform of an evenly symmetric interferogram contains only real components. The presence of dispersive elements and the possibility that the position of ZPD is not being sampled can result in an interferogram with signal samples that are not symmetric about ZPD. The resulting spectrum will contain both real and imaginary components and therefore a non-zero phase. Phase correction proceeds in two steps: the first characterizes the phase of the measured interferogram; the second removes the phase.

6.13.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observational building block.

SDS — **Spectrometer Detector Spectrum** This product contains the double-sided (low resolution) spectra for each spectrometer detector for each scan of the observation.

6.13.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** This output SDI contains phase-corrected interferograms for each spectrometer detector for each scan of the observational building block.

SDS — **Spectrometer Detector Spectrum** The set of double-sided spectra SDS that was provided as an input to this processing module will be modified such that its double-sided spectra will be phase-corrected.

6.13.4. Input Calibration Products

calBandEdges — **Spectrometer Band Edges** This optional calibration product contains the measured spectral band limits in units of cm^{-1} for each spectrometer detector.

6.13.5. How to use the *Phase Correction* Module

6.13.6. Control parameters

Users can set the following parameters of the Phase Correction module:

polyDegree — The optional parameter "polyDegree" sets the degree of the polynomial function that is used to fit the measured in-band phase. The parameter can be set to any positive integer. If "polyDegree" is set to unity for a linear fit, then phase correction can only shift ZPD, the point of symmetry of the interferogram, and, in the process, change where the interferogram is sampled. However, phase correction with a linear phase fit cannot change the shape of the interferogram. If "polyDegree" is set to values of two or higher then the underlying shape of the interferogram can change as well. The parameter has to be selected with care in the presence of significant noise. In this case, polynomials with a order greater than 4 can easily fit to random noise rather than a stable and repeatable instrumental phase. The default value is 4.

pcfSize — This optional parameter specifies the number of elements in the phase correction function (PCF) which is used to correct single-sided interferograms. A benefit of a larger PCF is that it may be able to correct for asymmetries at higher OPD. However, a larger PCF will also lead to a stronger reduction in spectral resolution because the convolution operation invalidates points at the interferogram edges. The convolution will invalidate a number of points on either side of the interferogram which is equal to half the value specified by "pcfSize". The default value is 256.

convolApodName — In order to avoid artifacts from a convolution with an array that has non-zero values at its edges, the PCF can be apodized. A range of apodizing functions is available. The

optional parameter "pcfApod" can be set to one of the values specified below. Care should be taken to select an apoizing function with values close to zero at its edges. It is a non-trivial decision to select the best apodizing function for phase correction. At the same time, the impact of the apodization function on the quality of the phase correction is limited. By default, no apodizing function is applied.

The following apodizing functions are available:

- "aNB_W_120", Weak Norton Beer
- "aNB_M_140", Medium Norton Beer
- "aP_141", Norton Beer P(α, p) function with α = -0.0325 and p = 0.3
- "aHM_150", Hamming
- "aHANN", Hanning
- "aNB_S_160", Strong Norton Beer
- "aBH_3_184", Blackman Harris, 3 terms
- "aE_195", Filler E(α) function with α = 0.22
- "aBH_4_221", Blackman Harris, 4 terms
- "aGAUSS", Gaussian
- "aNB_11" - "aNB_20", ten adjusted Norton Beer functions

6.13.7. Examples

Assuming a spectrometer detector interferogram product `sdi` and an observation context `obs`:

```
from herschel.spire.ia.pipeline.spec.ft import *

from herschel.spire.ia.pipeline.spec.phase import *

fourierTransform=FourierTransformTask()

phaseCorrection=PhaseCorrectionTask()

dsds = fourierTransform(sdi=sdi.copy(),
                        ftType="ds",
                        IA=True)

sdi = phaseCorrection(sdi=sdi,
                      sds=dsds,
                      polyDegree=4,
                      convolApodName="aNB_20",
                      pcfSize=127,
                      bandEdge=obs.calibration.spec.bandEdge)
```

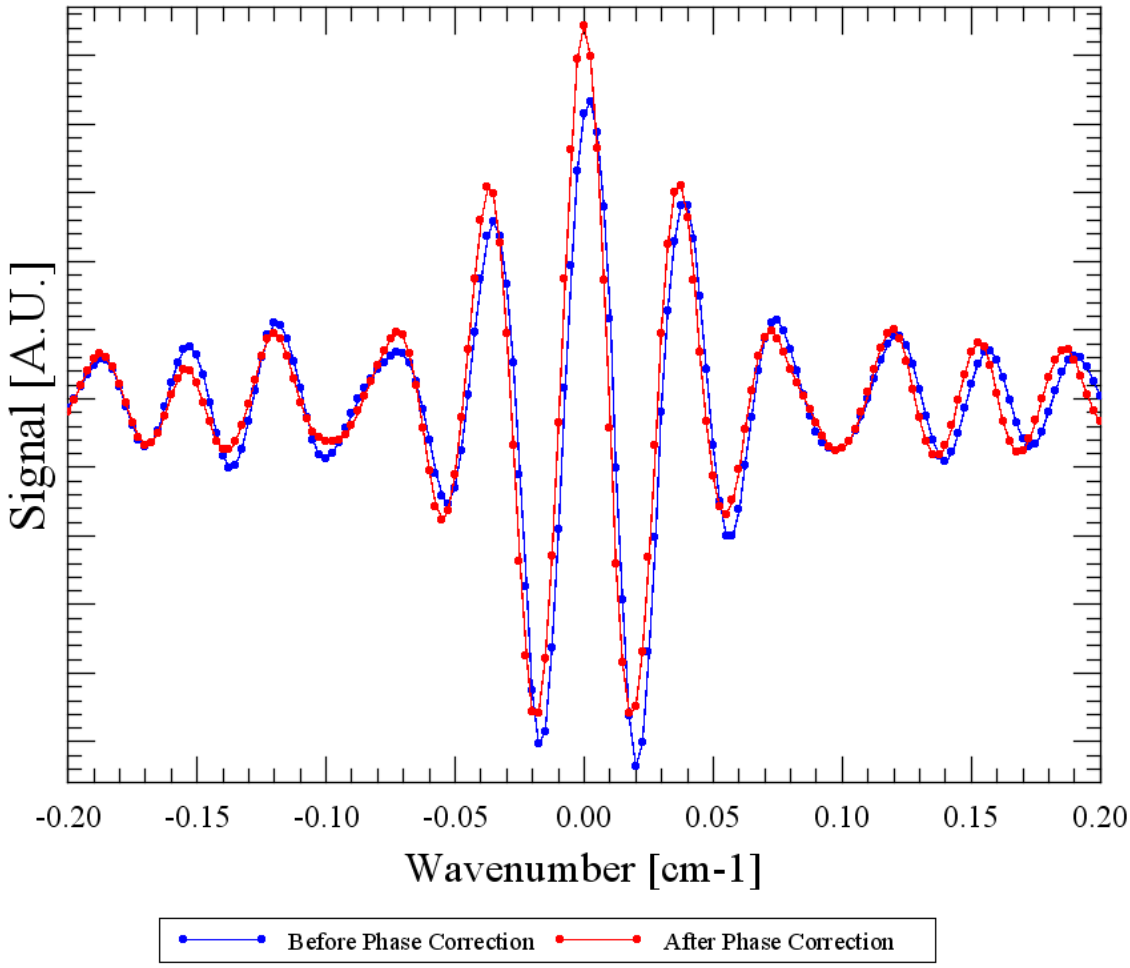

Figure 6.8. An example for phase-correction: The original interferogram (blue) shows a slight displacement from ZPD and asymmetries with respect to the center burst. Note how the first sidelobes at plus and minus 0.02 cm have different amplitudes. The phase-corrected interferogram (red) is well centered about ZPD and its first sidelobes have very similar amplitudes.

6.13.8. Error messages

error: No match for "xx" found in SDS. Cannot continue. This module will throw an exception if the low resolution SDS product does not contain a scan contained in the input SDI product.

error: No match for "xx" found in SDS. Cannot continue. This module will throw an exception if the low resolution SDS product does not contain a detector contained in the input SDI product.

error: Sdi has no "type" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "type" entry in its metadata.

error: Sdi has no "numScans" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "numScans" entry in its metadata.

error: Sdi has no "aot" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "aot" entry in its metadata.

error: Sdi has no "commandedResolution" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "commandedResolution" entry in its metadata.

error: Unknown Apodization Function. This module will throw an exception if the user-specified apodizing function is not available.

6.14. Applying an Apodization Function to an Interferogram

6.14.1. Module Description

The *Apodization* module multiplies the interferograms for each detector and scan of an observation building block with an analytically defined tapering function. The purpose of this processing step is to reduce the sidelobes of the instrumental line shape of the SPIRE spectrometer.

6.14.2. Input Data Products

SDI **Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.14.3. Output Data Products

SDI **Spectrometer Detector Interferogram** This product contains apodized interferograms for each spectrometer detector for each scan of the observational building block. If pre-apodization is selected then the output SDI product will contain double-sided interferograms. If post-apodization is selected then the output SDI product will contain samples for only those OPDs where OPD >= ZPD.

6.14.4. Input Calibration Data Products

No calibration data are needed.

6.14.5. How to use the *Apodization* Module

6.14.6. Parameter Options

apodType **Apodization Type** This parameter defines the type of apodization that is to be performed. If set to "ds", which stands for double-sided, the output SDI will contain samples before and after ZPD. If this parameter is set to "ss", which stands for single-sided, the output SDI will contain only those samples where OPD is >= ZPD.

apodFunctionName **Apodization Function Name** This optional parameter defines the apodizing function that is to be applied to interferograms in the input SDI product. By default, the apodization task uses the apodizing function aNB_15.

In addition to some long-standing apodizing functions, the SPIRE data processing environment offers ten functions that optimize the tradeoff between minimizing the secondary sidelobes of the instrumental line shape and reducing spectral resolution. The names of these apodizing functions indicate the loss of spectral resolution when applying the apodizing function. The factors run from 1.1 to 2.0 in steps of 0.1, indicating that the full width at half maximum of an unresolved line feature will increase by this factor. The names of the apodizing functions available for this task are as follows: aNB_11 ... aNB_20, where aNB stands for adjusted Norton-Beer. These apodizing functions are optimal in the sense of minimizing the secondary sidelobes of the instrumental line shape while causing the minimal reduction in spectral reso-

lution. Further details can be found in Naylor, D.A., and Tahic, M.K., "Apodizing functions for Fourier transform spectroscopy". J. Opt. Soc. Am. A 24, 3644-3648 (2007).

The following apodizing functions are available:

- "aHM_150", Hamming
- "aHANN", Hanning
- "aGAUSS", Gaussian
- "aNB_11" - "aNB_20", ten adjusted Norton Beer functions

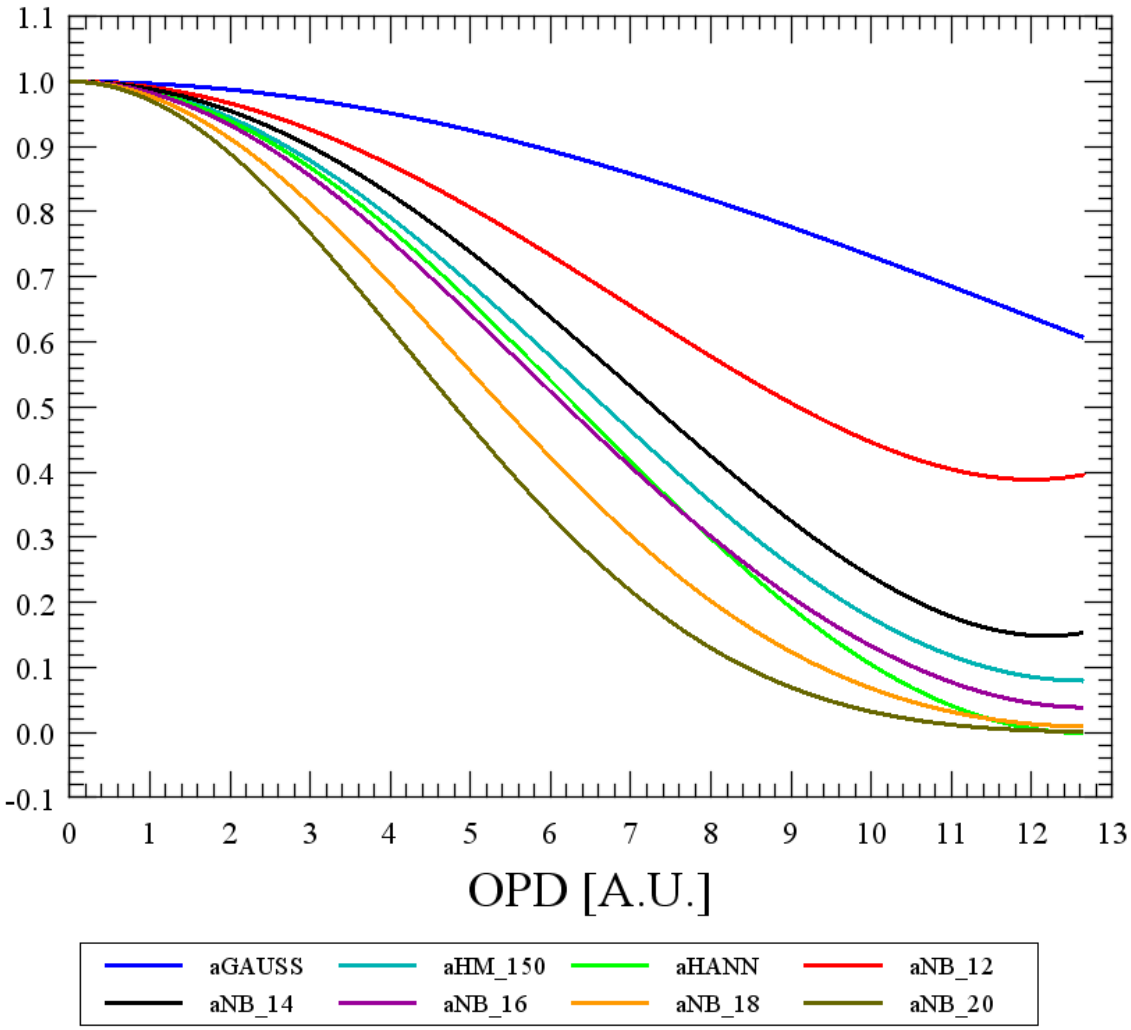

Figure 6.9. A subset of the available apodizing functions. The x-axis is OPD in arbitrary units. The apodizing functions will always be stretched to the maximum OPD. Some of the adjusted Norton-Beer functions are omitted for clarity.

6.14.7. Examples

The following is an example of pre-apodization, i.e. apodization of double-sided interferograms in the SDI prior to phase correction.

```
from herschel.spire.ia.pipeline.spec.apodize import ApodizeIfgms

apodizeIfgms=ApodizeIfgms()

presdi=apodizeIfgms(sdi=sdi,

                    apodType="ds",

                    apodFunctionName="aNB_15")
```

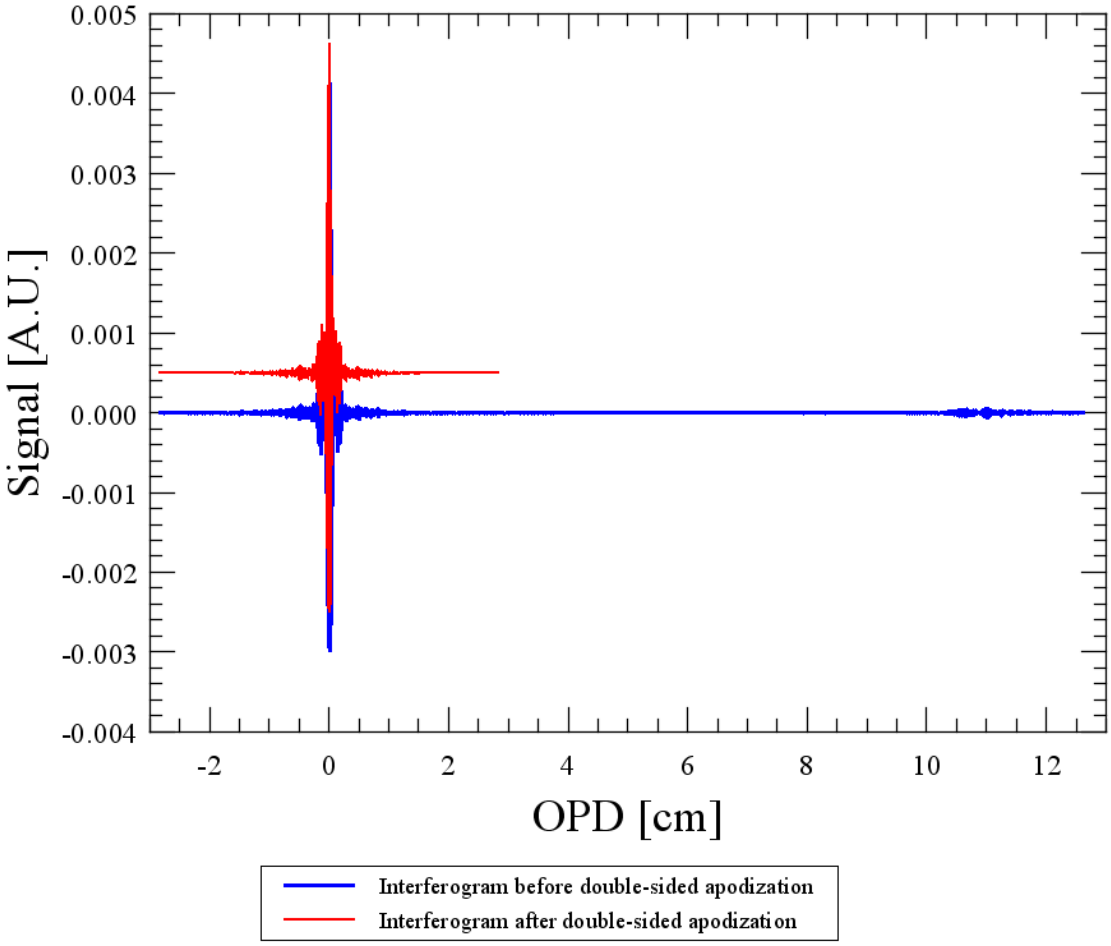

Figure 6.10. An example for double-sided apodization: The original interferogram (blue) and the apodized interferogram (red), offset by 0.0005 for clarity. The default apodizing function aNB_15 was used.

The following is an example of post-apodization, i.e. apodization of single-sided interferograms in the SDI after phase correction.

```
from herschel.spire.ia.pipeline.spec.apodize import ApodizeIfgms

apodizeIfgms=ApodizeIfgms()

sdi=apodizeIfgms(sdi=sdi,

                 apodType="ss",

                 apodFunctionName="aNB_15")
```

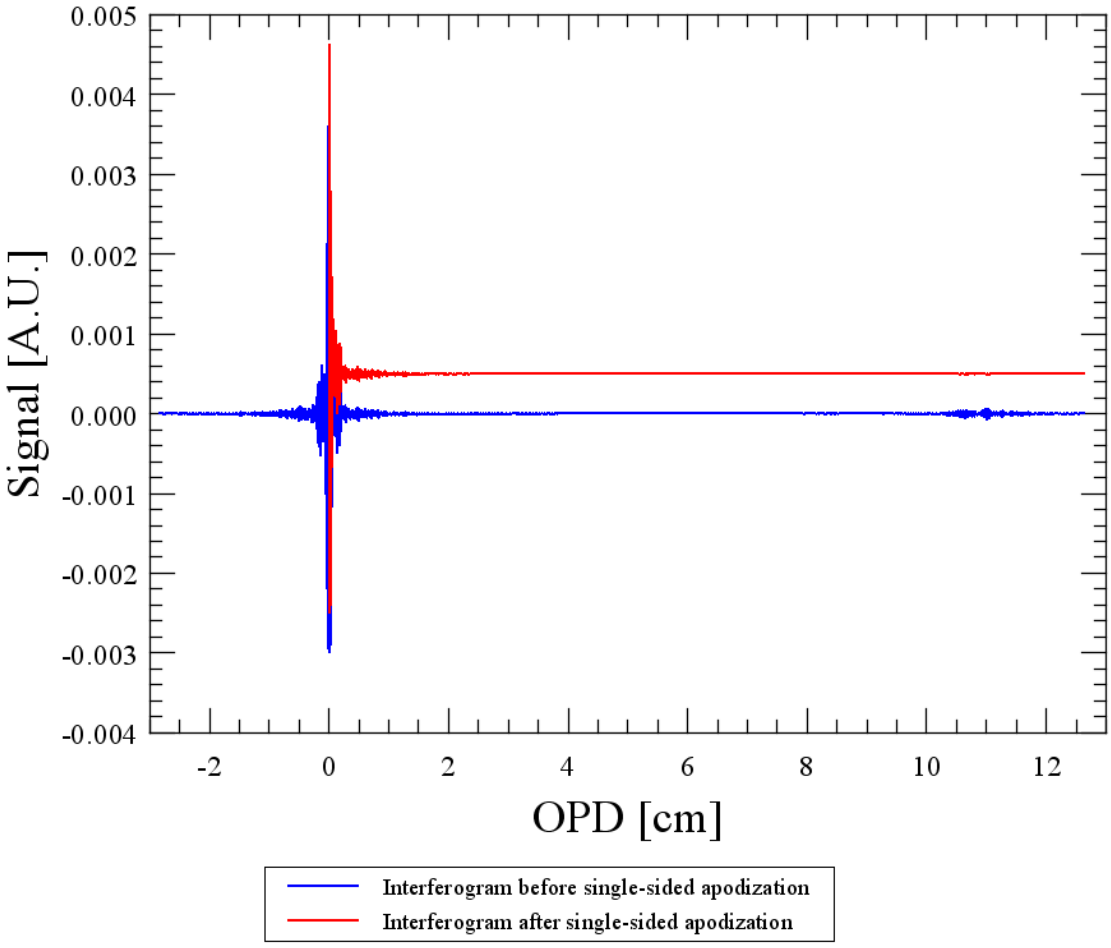

Figure 6.11. An example for single-sided apodization: The plot shows the original interferogram (blue) and the apodized interferogram (red), offset by 0.0005 for clarity. Note how the channel fringe feature at high OPD is deprecated in the apodized interferogram. The default apodizing function aNB_15 was used.

6.14.8. Error messages

error: Cannot get double sided portion: Range of OPD grid does not contain 0. error: Cannot get double sided portion: ZPD cannot be determined. This module will throw an exception if the interferogram does not contain ZPD.

error: Invalid argument Type. This module will throw an exception if the user sets the parameter apodType to a value other than "ss" or "ds".

error: Invalid argument name. This module will throw an exception if the user sets the parameter apodFunctionName to a value that refers to an apodizing function which is not available. A list of valid apodizing functions is printed to the console.

6.15. Producing Spectra from the Interferograms (Fourier Transform)

6.15.1. Module Description

The purpose of the *Fourier Transform* task is to transform the set of interferograms from a SPIRE spectrometer observational building block into a set of spectra. This processing task is capable of transforming both double-sided and single-sided interferograms. **Double-sided Transform.** For the double-sided transform, each interferogram in the SDI is examined and only the double-sided portion of the interferogram, where data are available between $-OPD_{max}$ and $+OPD_{max}$, is used to compute the resultant spectrum. The resultant spectra will contain both real and imaginary components. **Single-sided Transform.** For the single-sided transform, only those interferogram samples to one side of the position of zero path difference are considered. The spectra that result from the single-sided transform contain only real components.

6.15.2. Input Data Products

SDI **Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.15.3. Output Data Products

SDS **Spectrometer Detector Spectrum** The output SDS data product contains the calculated spectrum for each of the interferograms in the input SDI.

6.15.4. Input Calibration Products

No calibration data are needed.

6.15.5. How to use the *Fourier Transform* Module

6.15.6. Parameter Options

Users can set the following parameters of the Fourier Transform module:

ftType The user has two options for this parameter: "ds" for double-sided or "ss" for single-sided. If this parameter is set to "ds" then the portion of the input interferograms that is symmetric about the position of zero path difference will be used to compute the output spectra. If this parameter is set to "ss" then only the portion of the input interferograms that is greater than or equal to the position of zero path difference will be used to compute the output spectrum.

The data type of the flux columns in the output spectra depends on the value chosen for this parameter. The flux column will be of type `Complex1d` if "ftType" is set to doublesided. If "ftType" is set to "singlesided", then the flux column will be of type `Double1d` .

IA This Boolean parameter specifies whether zero-padding for the interferogram is enabled. If "IA" is set to "False" then the standard zero-padding is applied, as specified for the Standard Product Generation pipeline. If "IA" is set to "True" then the user has a choice of two other zero-padding options for interactive analysis.

ZP If the parameter "IA" is set to "True" then this Boolean parameter defines how interferograms are zero-padded. If "ZP" is set to "False" then no zeros are appended to the

interferogram. If "ZP" is set to "True" then zeros are appended to the interferograms to provide exactly 2^n data points to the Fast Fourier Transform routine.

6.15.7. Examples

The following are examples of the SPIRE spectrometer Fourier transform task.

```
from herschel.spire.ia.pipeline.spec.ft import *

fourierTransform=FourierTransformTask()

dsds=fourierTransform(sdi=sdi, ftType="ds", IA=False)

ssds=fourierTransform(sdi=sdi, ftType="ss", IA=True, ZP=True)
```

6.15.8. Error messages

error: Parameter ftType must be set. The parameter ftType is mandatory and must be set by the user.

error: Delta not a constant. The module throws an exception if the OPD grid is not equidistant. The result from this module would be incorrect if the calculation went ahead regardless.

error: Zero not found in array. Array does not have positve and negative values. The module throws an exception if there is no zero in the calculated OPD array.

6.16. Spectrum Flux Calibration

6.16.1. Module Description

The *Flux Conversion* module performs the absolute flux calibration for the SPIRE spectrometer pipeline. Based on an astronomical reference source, the values in the Spectrometer Detector Spectrum (SDS) product are converted from Volts per cm^{-1} to units relating to the flux of the source.

6.16.2. Input Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains a spectrum for each detector for each scan of the observational building block.

6.16.3. Output Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains a spectrum for each detector for each scan of the observational building block. The signal values are now in units that related directly to the observed astronomical source.

6.16.4. Input Calibration Products

SpecFluxConv — The **Spectrometer Flux Conversion** calibration product contains on a detector-by-detector and wavenumber-by-wavenumber basis the calibration factors to calibrate for absolute flux.

6.16.5. How to use the *Flux Conversion* Module

6.16.6. Parameter Options

The flux conversion module does not use any parameters that can be controlled by the user at run-time.

6.16.7. Examples

Assuming `ssds` is a spectrometer detector spectrum product with the signal in units of volts per cm^{-1} and `obs` an observation context:

```
from herschel.spire.ia.pipeline.spec.fluxconv import *

specFluxConversion = SpecFluxConversionTask()

ssds = specFluxConversion(sds=ssds, fluxConv=obs.calibration.spec.fluxConv)
```

6.16.8. Error messages

warning: "xx" not in cal product. No conversion performed. warning: Flux Conversion Calibration product does not contain an array that matches : "xx" No Conversion performed The module issues a warning if a detector that is present in the input SDS product has no entry in the calibration product. The flux conversion cannot be performed for that detector.

warning: Wavenumber grids not integer multiples. The module issues a warning if the wavenumber grid is not equidistant.

warning: Unable to interpolate [...] Will attempt extrapolation. The module issues a warning if flux conversion is attempted for a spectral range for which no calibration data are available.

6.17. Removing any Optical Crosstalk from the Spectra

6.17.1. Module Description

The *Remove Optical Crosstalk* module accounts for effects of the optical chain of the SPIRE instrument which lead to a different detector receiving radiation that should have been received by another detector.

6.17.2. Input Data Products

SDS	**Spectrometer Detector Spectrum** The input product contains spectra for each detector.

6.17.3. Output Data Products

SDS	**Spectrometer Detector Spectrum** The output product contains spectra for each detector.

6.17.4. Input Calibration Products

SpecOptCross — The **Spectrometer Optical Crosstalk** calibration product contains two matrices which describe the optical crosstalk for the detectors of the SLW and SSW detector array respectively.

6.17.5. How to use the *Remove Optical Crosstalk* Module

6.17.6. Parameter Options

The optical crosstalk correction module does not use any parameters that can be controlled by the user at run-time.

6.17.7. Examples

Assuming `ssds` is an SDS product and `obs` an observation context:

```
from herschel.spire.ia.pipeline.spec.optcross import *

specOptCrossCorrection = SpecOptCrossCorrectionTask()

ssds = specFluxConversion(sds=ssds, fluxConv=obs.calibration.spec.fluxConv)
```

6.17.8. Error messages

This module does not issue specific error or warning messages.

6.18. Averaging Spectra to Produce One Final Spectral Product

6.18.1. Module Description

The *Average Spectra* module computes, on a wavenumber-by-wavenumber basis for each spectrometer detector, the average and the uncertainty of the spectral intensities across all scans. The average is calculated as the arithmetic mean of the spectral components. The uncertainty is calculated as the standard deviation of the spectral components.

6.18.2. Input Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains spectra for each spectrometer detector for each scan of the observational building block.

6.18.3. Output Data Products

SDS — **Spectrometer Detector Spectrum** The output SDS data product contains the average spectra and uncertainties (as error) for each spectrometer detector. There will be one scan in the output SDS.

6.18.4. Input Calibration Data Products

No calibration data are needed.

6.18.5. How to use the *Spectral Averaging* Module

6.18.6. Parameter Options

Users can set the following parameters of the Spectral Averaging module:

dir — The optional "dir" parameter allows users to average spectra which were recorded while the mirror stage mechanism SMEC was traveling in a specific direction. If "dir" is set to its default value "False", then the output product will contain only one composite dataset with data from scans in either direction. If "dir" is set to "True", then the output product will contain two composite dataset with data averaged separately for forward and reverse scans.

outlierType — The averaging task will ignore outliers while computing the spectral average and standard deviation if this optional parameter is set to one of its allowed values "3Sigma", "STD", or "MAD". Statistical outliers are defined as values outside of an acceptable range. Acceptable ranges are defined as follows: the arithmetic mean plus or minus three times the standard deviation for "3Sigma"; the arithmetic mean plus or minus a threshold factor (see next keyword) times the standard deviation for "STD"; the median plus or minus a threshold factor (see next keyword) times the median absolute deviation for "MAD".

thresholdFactor — The user can define the threshold factor if the outlierType is "STD" or "MAD". The real value of the optional parameter "thresholdFactor" should be positive. The default value is 3.2905 in both cases.

6.18.7. Examples

Assuming `ssds` is an SDS product:

```
from herschel.spire.ia.pipeline.spec.average import *

averageSpectra=AverageSpectraTask()

asds=averageSpectra(ssds, dir=False, outlierType="STD", thresholdFactor=3.2905)
```

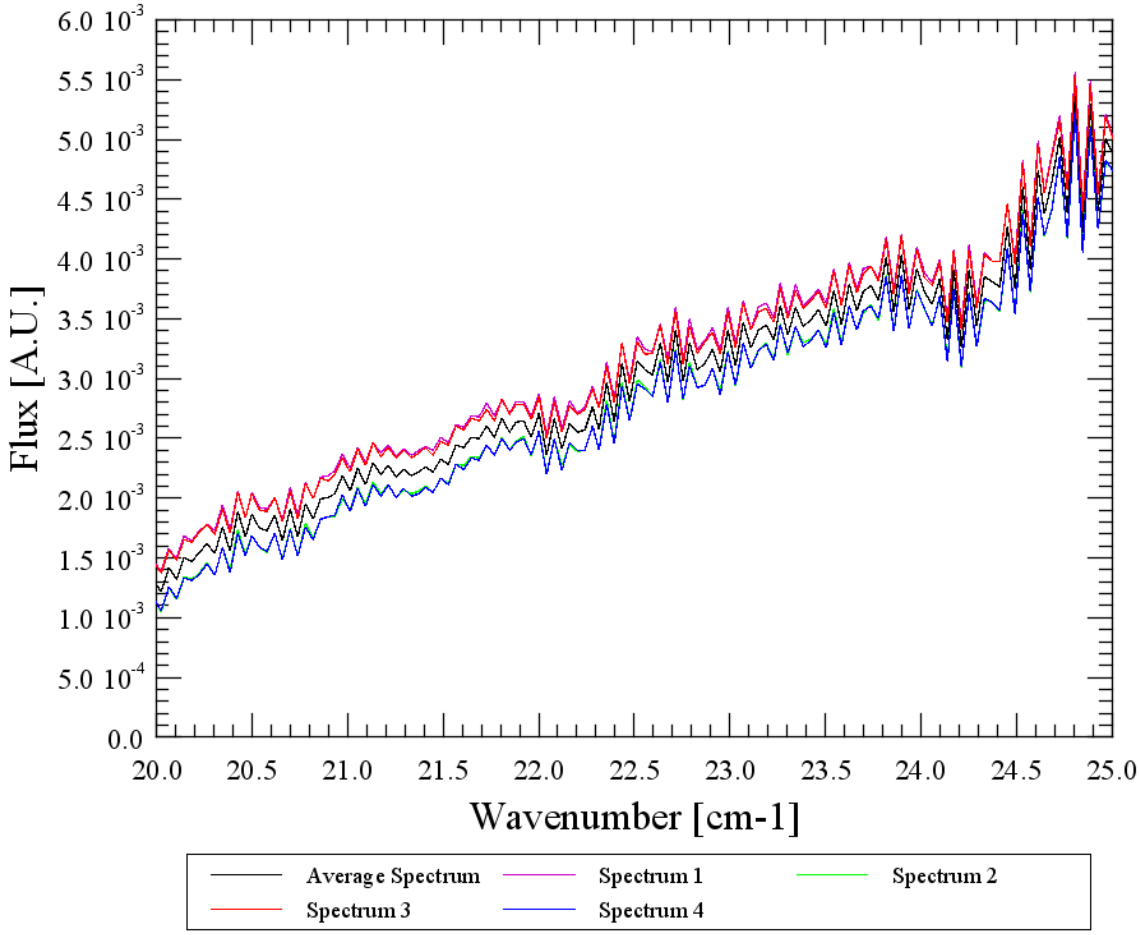

Figure 6.12. The spectra from four scans (magenta, green, red, blue) are averaged to yield one final spectrum (black).

6.18.8. Error messages

error: "xx" WN column length mismatch. error: "xx" Wavenumber column mismatch. For each detector, the task verifies that the wavenumber grids are identical for all scans. An exception will be thrown if they differ in length or by more than a small number which is below the precision of a variable of type `Float`.

error: Unsupported Flux type. The module throws an exception if the unit of the flux column is not supported.